

# Interactive Remote-Visualisation for Large Displays

Von der Fakultät Informatik, Elektrotechnik und  
Informationstechnik der Universität Stuttgart  
zur Erlangung der Würde eines  
Doktors der Naturwissenschaften (Dr. rer. nat.)  
genehmigte Abhandlung

Vorgelegt von

**Florian Frieß**

aus Geislingen an der Steige

Hauptberichter:	Prof. Dr. T. Ertl
Mitberichter:	Prof. Dr. D. Saupe
	Prof. Dr. T. Ropinski
Tag der mündlichen Prüfung:	08.07.2022

Visualisierungsinstitut  
der Universität Stuttgart

2022





This thesis is dedicated to Zoe for her patience, love, understanding and for the constant supply of cups of tea.



# ACKNOWLEDGMENTS

There is a long list of people to whom I owe thanks since they supported me and made this thesis possible. First and foremost, I want to thank my supervisor Thomas Ertl, not only for giving me the chance to work on my doctoral thesis at VISUS but also for giving me the freedom to pursue other projects I was interested in. I also want to thank Dietmar Saupe, as well as Timo Ropinski, for agreeing to act as referees for my thesis and for coming to my defense.

Of all the colleagues and co-authors at VISUS who worked on various projects with me, special thanks go to Michael Krone, who supervised my master thesis and also sparked my interest in pursuing a doctoral degree. Special thanks also go to my fellow colleagues and co-authors Christoph Müller, Valentin Bruder, Guido Reina, Steffen Frey, Karsten Schatz and Michael Becher, with whom I not only co-authored many papers but also had a lot of interesting and amusing discussions. Additionally, I want to thank the rest of my colleagues at VIS and VISUS, who always made it a great pleasure to work there. Furthermore, I am grateful to my project partners from the University of Konstanz, Dimitar Garkov, Johannes Häußler, Timon Kilian, Daniel Keim and Falk Schreiber, for the fruitful collaboration within the Project INF of the SFB-TRR 161. I am, of course, also grateful to my other co-authors from the University of Stuttgart: Matthias Braun, Patrick Gralka, Tobias Rau, Christoph Schulz, Mathias Landwehr, Katrin Scharnowski, Silvia Fademrecht, Tobias Kulschewski, Patrick C.F. Buchholz and Jurgen Pleiss.

My work was financially supported by the German Research Foundation (DFG) as project INF in the Collaborative Research Center SFB-TRR 161.

Last but not least, special thanks go to my parents, my wife Zoë and to my brother Marco all of whom supported me constantly throughout, and to Karen who took the time to read through my thesis. Countless thanks to all of you for always believing in me and encouraging me.



# TABLE OF CONTENTS

<b>Acknowledgments</b>	<b>v</b>
<b>Abstract</b>	<b>xiii</b>
<b>German Abstract – Zusammenfassung</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	3
1.2 Structure and Contribution . . . . .	4
<b>2 Fundamentals</b>	<b>7</b>
2.1 Visualisation and Rendering . . . . .	7
2.1.1 Visualisation Pipeline . . . . .	7
2.1.2 OpenGL Rendering Pipeline . . . . .	9
2.1.3 General Purpose Computation . . . . .	11
2.1.4 Large Displays . . . . .	11
2.1.5 Visual Acuity . . . . .	13
2.2 Biological Background . . . . .	14
2.3 Molecular Visualisation . . . . .	15
2.3.1 Simple Models . . . . .	15
2.3.2 Surface Models . . . . .	16
2.4 Image Transport . . . . .	18
2.4.1 Encoder . . . . .	19
2.4.2 Input Colour Format . . . . .	20
2.4.3 Intra Prediction . . . . .	21
2.4.4 Inter Prediction . . . . .	22
2.4.5 Frame Types . . . . .	22
2.4.6 Transformation, Quantisation and Entropy Coding . . . . .	23
2.4.7 Decoder . . . . .	24
2.5 Software Infrastructure . . . . .	25
2.5.1 Input Data . . . . .	25
2.5.2 The MegaMol Framework . . . . .	26
<b>3 Molecular Surfaces</b>	<b>29</b>
3.1 Molecular Surface Maps . . . . .	30
3.1.1 Method . . . . .	31
3.1.2 Molecular Surface Map Colouring . . . . .	35
3.1.3 Extension to Time-dependent Data . . . . .	37

3.1.4	Implementation Details . . . . .	37
3.1.5	Results and Discussion . . . . .	43
3.2	Clustering . . . . .	46
3.2.1	Method . . . . .	47
3.2.2	Results and Discussion . . . . .	53
3.3	Summary and Conclusion . . . . .	58
<b>4</b>	<b>Real-time High-resolution Visualisation System</b>	<b>61</b>
4.1	System Architecture . . . . .	66
4.1.1	Input Streams . . . . .	68
4.1.2	Encoding . . . . .	68
4.1.3	Network Transfer . . . . .	69
4.1.4	Rendering . . . . .	69
4.1.5	Configuration . . . . .	70
4.2	Implementation Details . . . . .	70
4.2.1	Media Foundation . . . . .	71
4.2.2	ASIO . . . . .	71
4.2.3	Screen Capturing . . . . .	72
4.2.4	Encoding . . . . .	73
4.2.5	Encoding Settings . . . . .	74
4.2.6	MPI Cluster Communicator . . . . .	75
4.2.7	Point-to-point Communication . . . . .	76
4.2.8	Decoding . . . . .	76
4.2.9	Display . . . . .	77
4.2.10	RTSP . . . . .	78
4.3	Test Setup . . . . .	78
4.4	Results and Discussion . . . . .	80
4.5	Summary and Conclusion . . . . .	89
<b>5</b>	<b>Algorithmic Optimisations</b>	<b>91</b>
5.1	Adaptive Encoding . . . . .	92
5.1.1	Method . . . . .	93
5.1.2	Implementation Details . . . . .	95
5.1.3	Results and Discussion . . . . .	99
5.2	Amortised Encoding . . . . .	105
5.2.1	Method . . . . .	107
5.2.2	Implementation Details . . . . .	109
5.2.3	Results and Discussion . . . . .	114
5.3	Summary and Conclusion . . . . .	121
<b>6</b>	<b>User-driven Optimisations</b>	<b>123</b>

## Contents

---

6.1	Tracking . . . . .	124
6.1.1	Optical Tracking . . . . .	124
6.1.2	HoloLens 2 . . . . .	125
6.2	Foveated Encoding . . . . .	126
6.2.1	Method . . . . .	127
6.2.2	Implementation Details . . . . .	129
6.2.3	Results and Discussion . . . . .	133
6.3	Stippling-based Encoding . . . . .	139
6.3.1	Method . . . . .	140
6.3.2	Implementation Details . . . . .	143
6.3.3	Results and Discussion . . . . .	146
6.4	Summary and Conclusion . . . . .	149
<b>7</b>	<b>Discussion and Conclusion</b>	<b>151</b>
	<b>Bibliography</b>	<b>163</b>





# LIST OF ABBREVIATIONS AND ACRONYMS

<b>AO</b>	Ambient Occlusion	<b>MPI</b>	Message Passing Interface
<b>API</b>	Application Programming Interface	<b>NAL</b>	Network Abstraction Layer
<b>CABAC</b>	Context-adaptive Arithmetic Coding	<b>NALU</b>	Network Abstraction Layer Unit
<b>CAVLC</b>	Context-adaptive Variable Length Coding	<b>OpenCL</b>	Open Computing Language
<b>CNN</b>	Convolutional Neural Network	<b>OpenGL</b>	Open Graphics Library
<b>CPU</b>	Central Processing Unit	<b>pixel</b>	Picture Element
<b>CUDA</b>	Compute Unified Device Architecture	<b>PPS</b>	Picture Parameter Sets
<b>DCT</b>	Discrete Cosine Transform	<b>QP</b>	Quantisation Parameter
<b>GLSL</b>	OpenGL shading language	<b>RAM</b>	Random Access Memory
<b>GOP</b>	Group of Pictures	<b>RGB</b>	Red, Green, Blue
<b>GPGPU</b>	General Purpose Computation on Graphics Processing Units	<b>RGBA</b>	Red, Green, Blue, Alpha
<b>GPU</b>	Graphics Processing Unit	<b>RMSD</b>	Root Mean Square Deviation
<b>GUI</b>	Graphical User Interface	<b>RTSP</b>	Real-Time Streaming Protocol
<b>GVF</b>	Gradient Vector Flow	<b>SAS</b>	Solvent Accessible Surface
<b>HVS</b>	Human Visual System	<b>SDK</b>	Software Development Kit
<b>IB</b>	Infini-Band	<b>SES</b>	Solvent Excluded Surface
<b>LCD</b>	Liquid Crystal Display	<b>SIMD</b>	Single Instruction, Multiple Data
<b>LED</b>	Light-emitting Diode Display	<b>SPS</b>	Sequence Parameter Sets
<b>MAR</b>	Minimum Angle of Resolution	<b>SSIM</b>	Structural Similarity Index
		<b>UPGMA</b>	Unweighted Pair Group with Arithmetic Mean
		<b>VLCs</b>	Variable-length Codes
		<b>VRAM</b>	Video Random Access Memory



## ABSTRACT

While visualisation often strives for abstraction, the interactive exploration of large scientific data sets, such as densely sampled 3D fields, massive particle data sets or molecular visualisations still benefits from rendering their graphical representation in large detail on high-resolution displays such as Powerwalls or tiled display walls. With the ever-growing size of data, and the increased availability of the aforementioned displays, collaboration becomes desirable in the sense of sharing this type of a visualisation running on one site in real time with another high-resolution display on a remote site. While most desktop computers – and in turn the visualisation software running on them – are alike, large high-resolution display setups are often unique, making use of multiple GPUs, a GPU cluster or only CPUs to drive the display. Therefore, particularly if the goal is the interactive scientific visualisation of large data sets, unique software might have to be written for a unique display and compute system.

Molecular visualisations are one application domain in which users would clearly benefit from being able to collaborate remotely, combining video and audio conference setups with the possibility of sharing high-resolution interactive visualisations. However, for large – often tiled – displays and image resolutions beyond 4K no obvious generic, let alone commercial, solution exists. While there are specialized solutions that support sharing the output of these displays, based on hardware-accelerated video encoding, these make compromises between quality and bandwidth. They either deliver a high quality image and therefore induce bandwidth requirements that cannot generally be met, or they uniformly decrease the quality to maintain adequate frame rates. However, in visualisation in particular, details are crucial in areas that are currently being investigated. Hence, an interactive remote-visualisation for high-resolution displays requires new methods that can run on different hardware setups and offer a high image quality while reducing the required bandwidth as much as possible.

In this dissertation, an innovative technique for rendering and comparing molecular surfaces as well as a novel system that supports interactive remote-visualisation, for molecular surfaces and other scientific visualisations, between different high-resolution displays is introduced and discussed. The rendering technique solves the view-dependency and occlusion of the three dimensional representation of the molecular surfaces by showing the topography and the physico-chemical properties of the surface in one single image. This also allows analysts to compare and cluster the images in order to understand the relationship structures, based on the idea that a visually similar surface implies a similarity in the function of the protein. The system presented in this dissertation uses a low latency pixel streaming approach, leveraging GPU-based video encoding and decoding to solve the aforementioned problems and to allow for interactive remote visualisations on large high-resolution displays. In addition to remote-visualisation the

system offers collaboration capabilities via bidirectional video and audio simultaneously. The system is based on the fact that, regardless of the underlying hardware setup, large displays share one property: they have a large (distributed or not) frame buffer to display coloured pixels. Consequently, this allows the users to collaborate between two sites that use different display walls with only a minimal delay. To address the bandwidth limitations, several methods have been developed and introduced which aim to reduce the required bandwidth and the end-to-end latency while still offering high image quality. The aim of these methods is to reduce the image quality and therefore the required bandwidth in regions that are not currently of interest to the users, while those that are of interest remain at a high quality. These methods can be categorised into algorithmic and user-driven optimisations to the remote visualisation pipeline. The user-driven optimisations make use of gaze tracking to adapt the quality of the encoding locally while the algorithmic optimisations use the content of the frames. Algorithmic optimisations include the usage of a convolutional neural network to detect regions of interest and adapt the encoding quality accordingly and a temporal downsampling prior to the encoding. These methods can also be combined, for example, foveated encoding may be combined with temporal downsampling to further reduce the required bandwidth and the latency.

Overall, this dissertation advances the state of the art by enabling the collaborative analysis of molecular and other scientific visualisations remotely at interactive frame rates without imposing bandwidth requirements that cannot generally be met.

# GERMAN ABSTRACT

## —ZUSAMMENFASSUNG—

Während die Visualisierung oft nach Abstraktion strebt, profitiert die interaktive Erkundung großer wissenschaftlicher Datensätze wie z.B. dicht abgetastete 3D-Felder, massive Partikel Datensätze oder molekulare Visualisierungen, immer noch von der Darstellung ihrer grafischen Repräsentation in großen Details auf hochauflösenden Displays wie Powerwalls oder gekachelten Displaywänden. Mit der stetig wachsenden Datenmenge und der zunehmenden Verfügbarkeit der zuvor erwähnten Displays ist eine Zusammenarbeit wünschenswert, um diese Art von Visualisierung, die an einem Standort läuft, in Echtzeit mit einem anderen hochauflösenden Display an einem anderen Standort zu teilen. Während die meisten Desktop-Computer—und damit auch die darauf laufende Visualisierungssoftware—gleich sind, sind große hochauflösende Bildschirmsysteme oft einzigartig, da sie mehrere GPUs, einen GPU-Cluster oder nur CPUs zur Steuerung des Bildschirms verwenden. Insbesondere wenn das Ziel die interaktive wissenschaftliche Visualisierung großer Datensätze ist, kann es daher nötig sein, eine spezielle Implementierung für ein dediziertes Anzeige- und Rechensystem zu realisieren.

Molekulare Visualisierungen sind eine Anwendungsdomain, in der die Nutzer eindeutig von der Möglichkeit der Zusammenarbeit aus der Ferne profitieren würden, indem sie Video- und Audiokonferenzen mit der Möglichkeit der gemeinsamen Nutzung hochauflösender interaktiver Visualisierungen kombinieren. Jedoch existiert keine offensichtlich generische und sicherlich keine kommerzielle Lösung für große—oft gekachelte—Displays und Bildauflösungen größer als 4K. Es gibt zwar spezialisierte Lösungen, die die gemeinsame Nutzung der Ausgabe dieser Bildschirme unterstützen und auf hardwarebeschleunigter Videokodierung basieren, aber diese gehen Kompromisse zwischen Qualität und Bandbreite ein. Sie liefern entweder ein hochwertiges Bild und führen daher zu Bandbreitenanforderungen, die im Allgemeinen nicht erfüllt werden können, oder sie verringern die Qualität gleichmäßig, um ein angemessene Bildrate aufrecht zu halten. Jedoch sind besonders in der Visualisierung, Details in Bereichen, die aktuell untersucht werden, von entscheidender Bedeutung. Daher benötigt eine interaktive Visualisierung von hochauflösenden Displays neue Methoden, welche auf verschiedenen Hardware-Setups laufen können und eine hohe Bildqualität bieten, während die erforderliche Bandbreite so weit wie möglich reduziert wird.

In dieser Dissertation werden eine neuartige Technik für Rendering und den Vergleich von Moleküloberflächen sowie ein neuartiges System, welches die interaktive Remote-Visualisierung für molekulare Oberflächen und andere wissenschaftliche Visualisierungen zwischen verschiedenen hochauflösenden Displays unterstützt, vorgestellt

und diskutiert. Die Rendering-Technik löst die Abhängigkeit der dreidimensionalen Darstellung der Moleküloberflächen von der Ansicht und der Verdeckung, indem sie die Topografie und die physikalisch-chemischen Eigenschaften der Oberfläche in einem einzigen Bild zeigt. Dies ermöglicht es Analysten auch, die Bilder zu vergleichen und zu clustern, um die Beziehungsstrukturen zu verstehen, basierend auf der Idee, dass eine visuell ähnliche Oberfläche eine Ähnlichkeit in der Funktion des Proteins impliziert. Das System, welches in dieser Dissertation vorgestellt wird, verwendet einen Ansatz für Pixel-Streaming mit niedriger Latenzzeit, der die GPU-basierte Videokodierung und -dekodierung nutzt, um die oben genannten Probleme zu lösen und interaktive Fernvisualisierungen auf großen hochauflösenden Bildschirmen zu ermöglichen. Zusätzlich zur Remote-Visualisierung bietet das System die Möglichkeit der simultanen Zusammenarbeit über bidirektionale Video- und Audioübertragung. Das System basiert auf der Tatsache, dass große Bildschirme unabhängig von der zugrundeliegenden Hardware eine Eigenschaft gemeinsam haben: Sie verfügen über einen großen (verteilten oder nicht verteilten) Framebuffer zur Anzeige farbiger Pixel. Dies ermöglicht es den Nutzern, zwischen zwei Standorten, die unterschiedliche Bildschirme verwenden, mit nur minimaler Verzögerung zusammenzuarbeiten. Um den Bandbreitenbeschränkungen zu begegnen, wurden mehrere Methoden entwickelt und eingeführt, die das Ziel haben, die erforderliche Bandbreite und die Ende-zu-Ende-Latenz zu reduzieren und gleichzeitig eine hohe Bildqualität zu bieten. Ziel dieser Methoden ist es, die Bildqualität und damit die benötigte Bandbreite in den Regionen zu reduzieren, die für die Nutzer derzeit nicht von Interesse sind, während die Regionen, die von Interesse sind, in hoher Qualität erhalten bleiben. Diese Methoden können in algorithmische und benutzergesteuerte Optimierungen der Fernvisualisierungspipeline unterteilt werden. Die benutzergesteuerten Optimierungen nutzen die Blickverfolgung, um die Qualität der Kodierung lokal anzupassen, während die algorithmischen Optimierungen den Inhalt der Bilder verwenden. Zu den algorithmischen Optimierungen gehören die Verwendung eines Convolutional Neural Networks zur Erkennung interessanter Regionen und zur entsprechenden Anpassung der Kodierungsqualität sowie ein zeitliches Downsampling vor der Kodierung. Diese Methoden können auch kombiniert werden, z.B. kann foveated Encoding mit zeitlichem Downsampling kombiniert werden, um die erforderliche Bandbreite und die Latenzzeit weiter zu reduzieren.

Insgesamt bringt diese Dissertation den Stand der Technik voran, indem eine gemeinsame Analyse molekularer und anderer wissenschaftlicher Visualisierungen aus der Ferne bei interaktiven Bildraten, ohne dass die Anforderungen an die Bandbreite im Allgemeinen nicht erfüllt werden können, ermöglicht wird.







## INTRODUCTION

Streaming technologies for working remotely and entertainment have become widely adopted in recent years. Video conferencing systems such as Skype, WebEx, Zoom, etc. support many users simultaneously but make compromises with respect to video and audio quality under heavy load. Streaming platforms like Netflix on the other hand can deliver 4K video resolution and high-quality audio, but only in one direction with no means of collaboration. Cloud gaming services deliver high quality content and allow for low-latency interaction, but for specific applications and desktop resolution only. Streaming technologies also play a role in science where remote visualisation is a common practice nowadays when it comes to displaying large amounts of data mainly resulting from simulations that are run on large-scale supercomputers. A typical scenario in remote visualisation is to render the simulation data on a cluster and send the encoded images to a client where they are decoded again. The main challenges when using this approach are maintaining a low latency, especially if the data should be interactively explorable, and keeping the required bandwidth below a certain limit. A natural approach to reduce latency, and in turn the required bandwidth, is to send less data, for instance, by strongly compressing the images with encoding settings that produce a low-quality output. However, in visualisation in particular, details are crucial in areas that are currently under investigation. In contrast, lower quality is sufficient in other areas that are mostly required for context.

Proteins are involved in almost all reactions inside an organism, therefore it is critical to understand their function. In addition to the wet lab experiments, Molecular Dynamics simulations are used to investigate the characteristics and properties of proteins. These experiments and simulations generate large amounts of data, which in turn require

new visualisations to enable the analysis of the results. Molecular surfaces are among the most widely used visual representations for the analysis of molecules, especially for the analysis of molecular dynamics simulations in biochemistry and structural biology. There are different surface models that show individual properties of the molecule. Further information can be mapped to the surface using colour, for example, to show physico-chemical properties of the underlying atoms. Additionally, the surfaces can be used to understand the relationship between protein structures, which remains a challenging yet important task for many application areas like drug design or biomedical research. This is based on the idea that a visually similar surface also implies a similarity in the function of the underlying protein [24, 159]. While these surface visualisations strive for abstraction, there is still a benefit in displaying these and other scientific visualisations in great detail on high-resolution displays. Firstly, this allows for an interactive exploration of the underlying data and secondly, this provides the opportunity for collaboration as multiple users can interact in front of the same large display.

Large displays are commonly wall-sized displays that are made of an array of liquid crystal displays (LCD displays), light-emitting diode displays (LED displays) or multiple video projectors and provide a high resolution image. These visualisation systems are typically rather unique in their setup of hardware and software. For instance, a system designed to mainly run browser-based information visualisation applications is typically designed to use as many graphic processing units (GPUs) as possible, each having as many video outputs as possible, in a single machine so that the application can run just like on a desktop computer. In contrast, if the system is designed to visualise large and complex scientific data sets, such as particle data or 3D flow fields, the system designers might want to add as much GPU power and memory as possible, hence opting for a GPU cluster. Others might even choose to use the aggregated power of the many central processing units (CPUs) used to run the simulation to compute the visualisation in situ. As more and more visualisation systems of this kind are installed, collaboration becomes desirable in the sense of sharing a visualisation running on one site in real time with another high-resolution display on a remote site while at the same time communicating via video and audio. Despite recent advances, visualising and analysing such simulations remotely at interactive frame rates remains a challenge for high-resolution set-ups. Many of the widely adopted solutions are aimed primarily at desktop or mobile device usage and do not offer support for large tiled high-resolution displays and image resolutions beyond 4k. Additionally they have to make compromises with respect to video quality under heavy load. Although there are specialized solutions that support large tiled high-resolution displays, they too make compromises between quality and bandwidth. They either deliver a high quality image and therefore induce bandwidth requirements that cannot generally be met, or they uniformly decrease the quality to maintain adequate frame rates.

## 1.1 Motivation

This dissertation presents techniques for the rendering of molecular surfaces and focuses on how to visualise them and other scientific visualisations remotely on high-resolution displays with a high image quality, while reducing the required bandwidth and the overall latency. While the aforementioned molecular surfaces are used for the analysis of molecular dynamics simulations, their three dimensional representation comes with the typical common problems inherent to three-dimensional visualisations such as view-dependency and occlusion. To solve these problems a novel approach that shows the topography and the physico-chemical properties of the surface in one single image is needed. This would enable the analyst to get a quick overview of the whole surface and identify interesting surface features for further investigation. Using large displays allows users to interactively compare multiple surfaces and offers the possibility for collaboration as wall-sized displays are suitable for collaborative work [73]. However, sharing the interactive visualisations in a remote setting, i. e. collaboration between multiple sites, each using their own large display, required new solutions for the hardware diversity as well as the bandwidth requirements.

While the aforementioned hardware diversity of large displays is irrelevant if only a few people are working in front of a single display wall, it becomes a problem if users want to collaborate between two sites with different display walls: software might have to be adapted in code to run on the other site, or the system driving the wall from a single computer with as many outputs as possible might not have the computational power or graphics memory to run a 3D visualisation of a large simulation data set. Even if the computational resources are not the problem, it might be prohibitively time-consuming to transfer the whole simulation trajectory between the sites. In order to support collaboration across sites using multiple wall-sized displays, a wide variety of systems, frameworks and libraries has been developed. They can be broadly categorised into pixel streaming approaches and web based approaches. The two, probably most widely known, collaboration environments for display clusters are the Scalable Adaptive Graphics Environment (SAGE) [124] and its successor, the Scalable Amplified Group Environment (SAGE2) [103, 123]. While SAGE is based on pixel streams, the newer SAGE2 is a browser-centric reimplementation around a central Node.js-based web server, which is responsible for distributing the shared content to different clients. Other pixel streaming methods include recent approaches by Biedert et al. [15] and Marrinan et al. [104]. Although systems that do not build on pixel streaming offer a wide range of collaboration techniques and support any client, as long as a recent browser is available, they do not offer the required performance to share the content of a large high-resolution display in real time. While systems that build on pixel streaming offer the required performance, they either do not support bidirectional audio and video or require code changes to applications in order to generate pixel streams. Additionally,

not all systems are capable of running on the different hardware setups mentioned above. Therefore a system that offers interactive remote-visualisation, supports collaboration with bidirectional audio and video streams and runs on different hardware setups, by leveraging all available hardware support, such as hardware video encoders and decoders, is needed.

This is, however, only the first step since the trade-off between image quality and required bandwidth still applies. As mentioned before the natural approach to reduce the required bandwidth is to send less data. The aforementioned systems use this approach to either deliver a high quality image at the cost of bandwidth requirements that cannot generally be met, or they uniformly decrease the quality to keep the required bandwidth in check. However, in visualisation in particular, details are crucial in areas that are currently under investigation, whereas for other areas that are mostly required for context a lower quality is sufficient. With techniques that uniformly adjust quality, some of the limited bandwidth is wasted on image areas outside of the user's region of interest. This effect is amplified for large displays since only a small part of the display is inside the user's region of interest. Even inside this region full details are not required everywhere, since humans typically cannot make out many details and colours outside a small foveal region around the fixation point. In contrast, near this point at the centre of the field of view, humans perceive the highest number of details and a broad range of colours. Strasburger et al. [154] provide an overview of different works on peripheral vision and pattern recognition. Therefore, novel approaches that reduce the latency and required bandwidth are needed to maintain interactivity, a high image quality and a low bandwidth even for large display resolutions. This enables collaborative work across sites without impacting the user experience, e.g. the loss of fine details from high compression.

## 1.2 Structure and Contribution

This section gives an overview of the structure of this dissertation and summarizes the contributions of the author to the discussed publications. All subsequent publications were co-authored by the author's PhD advisor Thomas Ertl. If not noted otherwise, the co-authors mentioned below were working at the Visualization Research Center of the University of Stuttgart (VISUS) at the time of the collaboration.

Chapter 2 provides an overview of the basics that are required for the rest of the dissertation. This includes the fundamentals of video encoding, scientific visualisation, rendering, and general purpose computations on graphics hardware as well as a short biological background. This first part of the chapter reviews the respective subjects based on previous work found in literature. The last part describes the MegaMol visualisation framework that was used as a prototyping platform for the implementations

discussed in chapter 3. MegaMol was originally designed and implemented by Sebastian Grottel [53] and gradually extended over the years [50]. The author of this dissertation contributed methods for rendering molecular surfaces, as well as other functionality.

Chapter 3 describes different methods focusing on improving the representation and comparisons of Molecular surfaces. In cooperation with Silvia Fademrecht, Tobias Kulschewski and Jürgen Pleiss from the Institute of Technical Biochemistry (ITB, University of Stuttgart), a method to mitigate the common problems like occlusion and view dependency, that arise from analysing the surface in three dimensions was developed [86] (as joint first author together with Michael Krone). The author of this dissertation contributed the conversion from genus  $n$  to genus zero as well as different map projections and surface colour modes. The visualisation described in section 3.1 shows the properties as well as the topography of the molecular surface in a single image solving the aforementioned problems. Section 3.2 describes image based clustering for these maps in order to help with the analysis of bigger datasets. The approach uses a Convolutional Neural Network and a hierarchical image-based clustering in order to find proteins with a similar structure and surface properties [141] (as joint first author together with Karsten Schatz). An extended version was published in *Computers & Graphics* [140] (as joint first author together with Karsten Schatz). The author of this dissertation contributed the hierarchical clustering approach as well as the implementation of the Molecular Surface Maps, described in [86].

Chapter 4 describes the baseline framework, upon which the extensions described in chapter 5 and 6 build, and reviews its strengths and weaknesses [46]. This includes a technical description of the implemented system as well as performance evaluations. Although the system is tailored towards large displays and high resolutions, it also works for workstations, mobile devices, such as laptops, and desktop computers. It offloads parallelisable computations to the GPU and the encoding and decoding to the dedicated hardware encoder and decoder chips, which also reside on the GPU.

In chapter 5, two methods that improve the framework introduced in chapter 4 are discussed. Again this includes technical descriptions of the implemented methods as well as performance evaluations. Both build upon the baseline framework and make algorithmic changes to the encoder pipeline in order to reduce the required bandwidth while keeping the overall image quality as high as possible. The first, described in section 5.1, uses a Convolutional Neural Network (CNN) to detect regions of the image that contained finer structures [45]. These regions are encoded with less compression while other, less interesting regions, are compressed more in order to reduce the required bandwidth. Section 5.2 describes the second approach that is based on amortised rendering techniques [43]. It uses spatial subsampling with temporal coherence to reduce the resolution of frames before encoding, which leads to a reduced encoding latency and a reduction in the required bandwidth.

Chapter 6 discusses further methods that aim to reduce the required bandwidth and the end-to-end latency. As in the two previous chapters this includes technical descriptions and performance evaluations of the methods. All methods make use of user data, such as gaze tracking, to modify the quality of the encoding. Again they extend the baseline framework. Section 6.2 discusses foveated encoding, which adapts the quantisation of the macroblocks used by the encoder, based on regions of interest, i. e. regions that are actively looked at by users. This improves the image quality locally while reducing the required bandwidth due to the high compression of the remaining parts. It was awarded Best Paper at the Large Data Analysis and Visualization (LDAV) conference and thus, the authors were invited to contribute the paper to the journal *IEEE Transactions on visualization and Computer Graphics* [44]. Section 6.3 describes a different approach to foveated encoding that reduces the size of the image by using a sampling pattern prior to the encoding. This pattern is precomputed based on the visual acuity fall-off and the number of samples is reduced based on the distance to the gaze point.

Chapter 7 concludes the dissertation by discussing the opportunities and benefits of the presented methods and examines their usability in solving actual research questions. The chapter concludes with a discussion of promising directions for future research based on the results of this dissertation.

## FUNDAMENTALS

This chapter gives an overview of the fundamentals on which this thesis is built. Firstly, the basics of scientific visualisation are explained. Secondly, the required technical foundations for rendering followed by a short introduction to molecular visualisation are discussed. After this, a brief overview of large high-resolution displays and image transport is given. Finally, the software infrastructure used for the generating and rendering of the molecular surfaces is detailed.

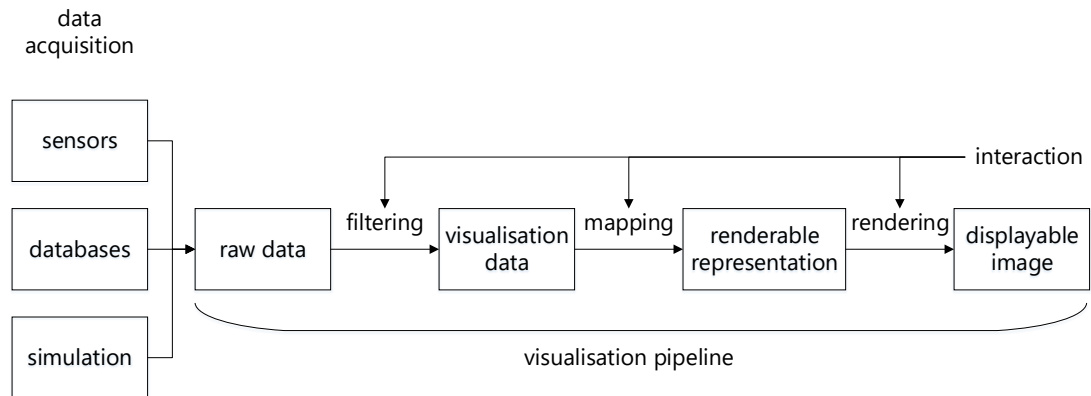
### 2.1 Visualisation and Rendering

The process of generating a graphical representation of numerical data is called visualisation. Its goal is to generate an abstract and meaningful image that allows to see and analyse such data. This data can originate from various sources, such as sensors or simulations and can be either spatial or non-spatial. Following this, visualisation can be categorised into scientific visualisation, which comprises of methods for spatial data, and information visualisation, which comprises of methods for non-spatial data.

#### 2.1.1 Visualisation Pipeline

Although visualisation can be categorised into two different types, scientific and information visualisation, the visualisation pipeline is the same for both as it is independent of the type of input data. The pipeline is a reference model that describes the visualisation process from the input data to the displayed image. Despite the fact that multiple



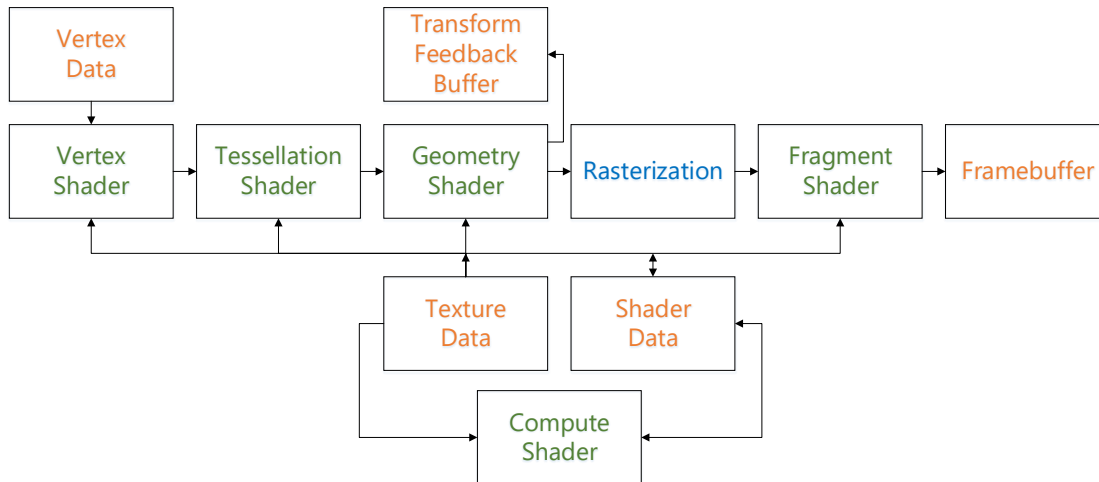


**Figure 2.1** – The visualisation pipeline as described by Weiskopf [168], including the data acquisition.

different variations of this pipeline have been proposed, most visualisation applications follow the common scheme given by Weiskopf [168], which is outlined in Figure 2.1.

The data acquisition step is not part of the visualisation pipeline but it provides the raw data, which is the input of the visualisation pipeline. This data can be acquired from a wide variety of sources, such as simulations, sensors or databases. The first stage of the pipeline is the usage of filters to transform the raw data into visualisation data. These filters include the removal, interpolation, quantisation or classification of certain data values. In this stage multiple filters can be applied successively, but there is also data which requires no filtering. In the second stage the mapping is applied, which transforms the visualisation data into renderable representation. This representation has an extension in space and time and contains attributes such as colour or geometry. This is achieved by mapping the potentially abstract visualisation data to graphical primitives, for example, by using a transfer function in order to assign colour values to temperature values. In the third and final stage, a displayable image is generated and presented to the user. As shown in Figure 2.1 these three stages – filtering, mapping, and rendering – can involve user interaction, which is a vital part of visualisation as it allows the exploration of the parameter space. This interaction can include the interpolation function for filtering, the selection of a transfer function during the mapping, or the adjustment of the camera for the rendering. Based on the interaction, the final image might show different aspects of the data and can lead to new insights.





**Figure 2.2** — The simplified OpenGL 4.6 pipeline with the programmable shaders in green, the rasterization, which converts the vertex data to fragment data, in blue and the data buffers in orange.

### 2.1.2 OpenGL Rendering Pipeline

The final stage of the aforementioned visualisation pipeline, the rendering, often uses a hardware-accelerated graphics Application Programming Interface (API) such as OpenGL or Direct3D for the interactive visualisation of three-dimensional data. This section introduces this concept by explaining the rendering pipeline used by modern OpenGL.

In computer graphics a three-dimensional object in a scene is represented by polygonal primitives. These primitives consist of vertices which have a position and additional attributes like colour, texture coordinates or normal vectors for lighting. The image generation follows a pipeline which transforms the primitives into the eye space of a virtual camera, then rasterizes and shades them. On a modern Graphics Processing Unit (GPU), almost all stages of this pipeline are programmable with only a few fixed-function operations. Since the computations for image generation are parallelisable, GPUs are built to make use of the Single Instruction, Multiple Data (SIMD) principle. That is, the same calculations are executed for all data points. The simplified overview (see Figure 2.2) of the OpenGL 4.6 pipeline [145] shows the four programmable stages: the *Vertex Shader*, the *Tessellation Shader*, the *Geometry Shader* and the *Fragment Shader*, as well as the, also programmable, *Compute Shader* which is not part of the standard pipeline. All of these shaders can be programmed using the OpenGL Shading Language (GLSL), can read data from texture memory, can read and write to GPU memory and use

a set of built-in input and output values to transport data between the stages. Additional input and output values for each shader can also be defined by the user. For Direct3D the pipeline is similar but the names and the programming language used for the shader are different.

The input to the *Vertex Shader* is the vertex data which consists of the vertices of the primitives in the scene and their respective attributes, such as position, colour and normal vector. The shader can be used to modify the input values, usually by applying the model-view-projection matrix to each vertex. The model matrix transforms the vertex from the initial object coordinates into world coordinates. With the view matrix the vertex is then transformed from world coordinates into camera coordinates. And finally, the projection matrix transforms from the camera coordinates into clip coordinates.

The next stage, the tessellation stage, is optional and can be used to subdivide the input polygonal patches. There are two shaders, both of which can be programmed, and the fixed-function tessellator, which is called in between the two. The first is the *Tessellation Control Shader*, which is used to define the level of the subdivision and therefore controls the number of resulting primitives. Then the tessellator generates the new primitives and forwards them to the second shader, the *Tessellation Evaluation Shader*. This shader is used to define the position and other attributes of the resulting vertices, similar to the *Vertex Shader*.

The, also optional, *Geometry Shader*, is executed for each primitive and can be used to generate new primitives, similar to the tessellation stage. Additionally it can be used to remove geometry by culling, i.e. the removal of unnecessary geometry. Compared to the tessellation it offers greater flexibility but is limited by the number of primitives that can be emitted and usually slower.

After the geometry stage the final vertex positions can be stored in the *Transform Feedback Buffer* using the transform feedback. This allows to download the data into main memory for further processing by the CPU or to store it to permanent memory. The next two stages in the pipeline are not programmable: the clipping, which removes primitives that are outside of the view frustum of the camera, and the rasterization, which transforms the vertex data into fragment data.

The rasterization results in fragments which have a position and a depth as well as interpolated additional attributes, e.g. colour, normal and texture coordinates. These fragments are the input of the *Fragment Shader*, which is used to modify the output values, i.e. the colour and the depth per fragment. The colour is typically stored as red, green, blue and alpha values (RGBA format). Before the final image is stored in the output buffer, optional non-programmable per-fragment operations such as blending can be executed. The output buffer can either be the *Framebuffer*, in case of direct rendering, or a buffer object in GPU memory in case of offscreen rendering.

### 2.1.3 General Purpose Computation

While GPUs are traditionally optimized for triangle rendering, they can also be used for advanced rendering methods like ray casting for direct volume rendering (see e.g. [23]) or other general purpose computations. Due to their higher degree of parallelism compared to CPUs they can achieve much lower run times for many parallelisable algorithms. Therefore, General Purpose Computation on GPUs (GPGPU) is often used to accelerate such algorithms. OpenGL and Direct3D offer a programmable *Compute Shader*, which is not part of the pipeline and therefore more flexible compared to the same implementation using the programmable shaders of the pipeline. The Open Computing Language (OpenCL) supports parallel computations on GPUs, CPUs and other processors, while NVIDIAs Compute Unified Device Architecture (CUDA) programming model allows the implementation of GPU programmes using a C-like programming language. Several of the algorithms discussed in Chapter 4-6 were implemented using Direct3D 11.1 compute shaders.

All of the aforementioned models follow a common scheme. The input data has to be transferred from the host (CPU) memory to the device (GPU) memory. Then the shader, or kernel (OpenCL/CUDA) is executed on the GPU. The threads on the device are organised into thread blocks and each block is executed concurrently. Each thread is assigned a unique ID, which can be used to access the input or output data, and executes the shader or kernel. After the computation is finished the output can either be transferred back to host memory or written to device memory. In general, the memory transfer between host and device should be kept to a minimum, as it often leads to a bottleneck due to the relatively low bandwidth.

### 2.1.4 Large Displays

Large tiled displays are currently either made out of an array of liquid crystal displays (LCD displays), light-emitting diode displays (LED displays) or multiple video projectors. While using a setup that consists of multiple projectors (cf. Figure 2.3) has the advantage of a bezel-less image, it requires frequent calibration. Additionally, this setup allows to use stereo projection. However, it is more expensive than using multiple LCDs or LEDs and recently there are bezel-less displays that also offer a high resolution. Therefore, only small tiled displays can be driven by a single computer and run arbitrary existing visualisation programmes. For larger displays multiple display nodes are required in order to provide enough connectors for the projectors or displays. Optionally, a graphics cluster can be used that renders the frames and transmits them to the display nodes, which can also be part of that graphics cluster. This requires special software which is often custom-made for the cluster it runs on, for example, by extending existing in-house visualisation programmes.



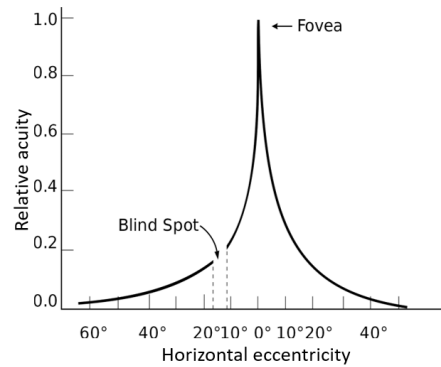
**Figure 2.3** — The projector array of the stereo capable large high-resolution display at VISUS<sup>1</sup>. The ten projectors are divided up into two groups, five project the left and the other five the right eye of the stereo image. They are placed on a metal framework that allows to move each projector individually in order to align them to get an image without holes. © VISUS

In order to display content on these displays, a variety of parallel rendering frameworks and middleware have been suggested which aim to help developers in writing software for graphics clusters or extending the available programmes. The Cross Platform Cluster Graphics Library (CGLX) [34] is based on OpenGL and follows a synchronised execution pattern by running an instance of the same application on every node. It allows users to adapt existing OpenGL-based applications or develop new applications for clusters such as tiled displays. Equalizer [37] and OmegaLib [39] provide a parallel OpenGL-based middleware with a large set of features to support a wide variety of research and industry applications. Both support systems ranging from large display walls to CAVEs. Chromium [65] also builds on OpenGL but provides distributed rendering without source code access, by intercepting the OpenGL commands and distributing them to rendering servers that execute it.

As previously mentioned, the rendering is distributed to multiple computers (nodes), which either use GPUs or CPUs for rendering, and might be part of a cluster. Each node only renders a part of the overall image which is then either displayed directly, if the cluster nodes are directly connected to the displays or projectors, or forwarded to display nodes. In both cases the connection between the nodes is usually based on high-speed network interconnects like Infini-Band (IB), which make distributing high-

<sup>1</sup> Visualization Research Center of the University of Stuttgart (VISUS) <https://www.visus.uni-stuttgart.de/institut/visualisierungslabor/> (last accessed 22/04/2022)

**Figure 2.4** — The relative acuity of the left human eye (horizontal section) in degrees from the fovea, as described by Hunziker [67]. As can be seen, the area where humans can perceive sharp, colourful details, is in the small foveal region around the centre of the field of view, while information in the periphery is perceived blurred and colourless since the relative acuity decreases rapidly.



resolution imagery in real-time a feasible solution. Communication between the nodes uses the Message Passing Interface (MPI), the de-facto standard for communication in HPC clusters. MPI provides a portable, cross platform, API and offers the the additional advantage of having specialised implementations for the aforementioned high-speed network technologies like IB. These implementations bypass the whole TCP/IP stack by using IB verbs, thus greatly reducing network latency and increasing bandwidth. This allows the transmission of frames without compression between the nodes as the available bandwidth is sufficient and the compression would increase the latency.

### 2.1.5 Visual Acuity

The ability of a person to recognize and distinguish small details is usually referred to as *visual acuity*, see Figure 2.4. There is a large body of work that attributes to the human eye a fall-off in visual acuity towards the periphery. This means humans typically cannot discern many details and colours outside a small foveal region around the fixation point. In contrast, near this point at the centre of the field of view, humans perceive the highest number of details and a broad range of colours. Strasburger et al. [154] give an overview on different works on peripheral vision and pattern recognition.

The fall-off can be approximated by different functions. The minimum angle of resolution (MAR) is the result of inverting the acuity and it can be approximated by a linear model [169]:  $MAR = \omega_0 + m * e$ .  $\omega_0$  denotes the smallest resolvable angle,  $e$  the eccentricity in degrees and  $m$  denotes the slope. The MAR model has been confirmed to match performance results in low-level vision tasks as well as anatomical features of the eye. This model has been used in numerous works in computer graphics and visualisation that implement foveated rendering methods [54, 153, 161]. Bruder et al. [23] used a 2D Gaussian function that depends on the screen resolution, size, approximated viewing distance, and estimated photoreceptor distribution.

## 2.2 Biological Background

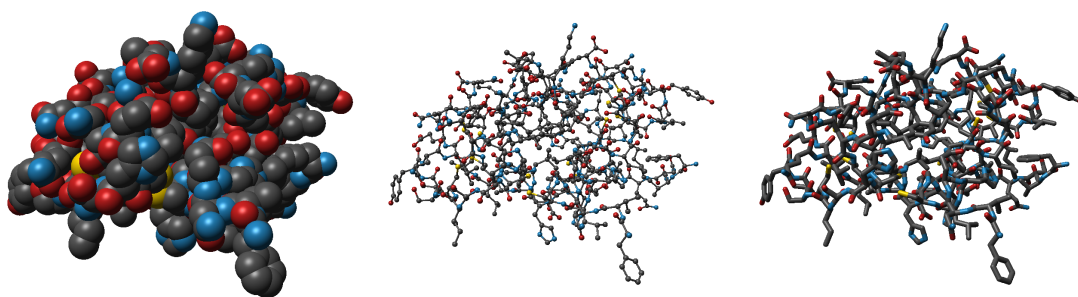
### Parts of this section have been published in:

- K. Schatz, F. Frieß, M. Schäfer, P. C. F. Buchholz, J. Pleiss, T. Ertl, and M. Krone. Analyzing the similarity of protein domains by clustering Molecular Surface Maps. *Computers & Graphics*, 99:114–127, 2021. [140]

Proteins are macromolecules that consist of a single or multiple chains of amino acids. A typical chain has 100–500 amino acids, each one consisting of about 10–27 atoms. All amino acids have a basic structure in common, the *backbone* part, which consists of a central carbon atom called  $C\alpha$ , an amino group ( $NH_2$ ), a carboxyl group ( $COOH$ ), and a hydrogen (H). Besides the backbone, each amino acid has a so-called *side chain* that determines the individual chemical properties of this amino acid. In proteins, amino acids form chains via peptide bonds. These are characteristic covalent bonds that link the amino group of one amino acid to the carboxyl group of the next one. That is, the chains of a protein have one end that is connected with an amino group (N-terminus) and one end that is connected with a carboxyl group (C-terminus). For more details, please refer to the book by Berg et al. [13]. When forming a protein, the chains fold into the so-called *tertiary structure*, the energetically most favorable 3D conformation, which is held together mainly by hydrogen bonds. It can be interesting to investigate specific domains of these chains, as they may have followed different evolutionary paths. Domains are parts of a chain that provide a specific function in the protein. Proteins serve many different tasks in the bodies of all living creatures as well as in a wide variety of industrial and medical applications, thus the analysis of their function and evolution is of great interest.

Starting from the amino acid sequence of a given protein, many predictions are possible. Using only this sequence, the tertiary structure and even the function of the complex can be inferred. At the widely-known CASP (Critical Assessment of protein Structure Prediction) experiment that takes place biennially, a benchmark for all methods that try to predict the tertiary structure is often published [109]. Proteins with similar sequences often also share an evolutionary relationship and have a similar function. To find proteins with similar sequences in a large data base, search methods such as BLAST [6] (Basic Local Alignment Search Tool) have been developed. To express and to display the aforementioned evolutionary relationship and similarity, biologists typically use phylogenetic trees [40]. These trees are often rendered as cladograms (to display only relationships) or dendrograms (to additionally encode distance). Dendrograms can be used to quickly find similar organisms in terms of structure, shape, or function, depending on the chosen comparison operator. However, proteins with vastly different sequences can have a similar 3D structure due to the folding of the chains. Therefore,





**Figure 2.5** — Simple atomistic models of the protein with the PDB ID 1RWE coloured by element: carbon: gray, nitrogen: blue, oxygen: red, sulphur: yellow. From left to right: the Space-filling model, the Ball-and-Stick model and the Stick model.

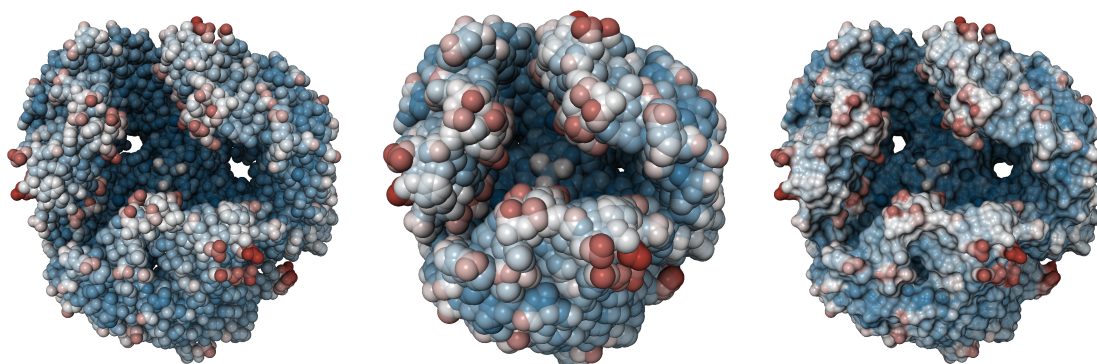
it is possible that their function is also similar. This fact can be exploited for drug discovery [78], or to reveal distant evolutionary relationships between organisms [80]. Conversely, some proteins can fold into different 3D structures under certain circumstances. These misfolded proteins (called *prions*) can be dysfunctional, causing severe diseases such as Alzheimer's or Creutzfeldt Jakob. Therefore, the sequence of a protein alone, is not always sufficient to faithfully analyse the function of a protein.

## 2.3 Molecular Visualisation

The visualisation of proteins or other molecules is an important tool to analyse their structure as well as their function. Therefore, an abundance of models have been devised in order to highlight different aspects of the molecules. They can be categorised into atomistic models and abstract models [84]. The atomistic models depict the position of the atoms that make up the protein and can be further classified into simple models and surface models. The simple models show all individual atoms while the surface models depict the interface between a molecule and its environment. The abstract models can be used to highlight a specific feature of a molecule, which might be, partially, obstructed in an atomistic representation. The following two subsections give a short overview of the simple and surface models that are relevant for the methods described in chapter 3.

### 2.3.1 Simple Models

The first simple model is the Space-filling model that represents each atom with a sphere, see leftmost image in Figure 2.5. The centre of the sphere is at the position of the atom and the radius of the sphere is proportional to the radius of the respective



**Figure 2.6** — Surface models of the protein with the PDB ID 1AF6 coloured by temperature factor (or B-factor), which is an indicator of the flexibility of the protein. From left to right: the van-der-Waals surface, the SAS and the SES.

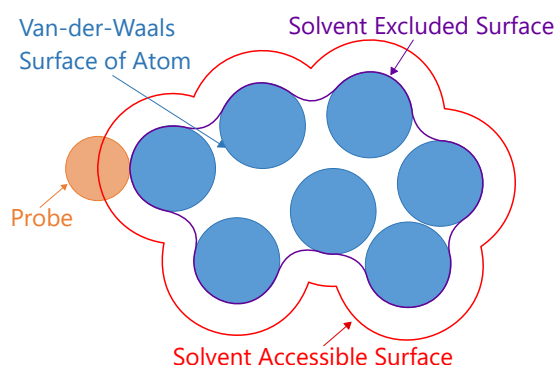
atom. The spheres for atoms that are bound to each other highly overlap, while the spheres on non-bound atoms do not, or only marginally, overlap. The second simple model is the Ball-and-Stick model which illustrates the atomic bonds as well as the positions of the atoms, see centre image in Figure 2.5. The atoms are again depicted by spheres, but their radius is much smaller than the atomic radius so that the bonds are still visible. The third simple model is the Stick model, which is a simpler version of the Ball-and-Stick model as the atoms are not depicted as spheres, see rightmost image in Figure 2.5. Bonds are depicted by thin cylinders and are coloured halfway according to the element of the atom on each side.

These models typically consist of three different geometric primitives. The atoms are represented by spheres, while the bonds are either represented by lines or cylinders. As described in section 2.1.2, three-dimensional objects, such as spheres and cylinders, are traditionally tessellated, i.e. subdivided into triangles, for GPU-accelerated rendering. Nowadays spheres are rendered using a mixture of rasterisation and raytracing. For each object an imposter shape, usually a quad, is rasterised and raytraced in the fragment shader. This reduces the computation time substantially, while producing pixel accurate images without the usual artefacts introduced by tessellation.

### 2.3.2 Surface Models

The van-der-Waals surface is a special type of the Space-filling model, see left image in Figure 2.6. Instead of using a radius for the spheres that is proportional to the atomic radius it uses a radius which is equivalent to the van-der-Waals radius of the atom. The van-der-Waals surface is the union of all spheres and shows the volume, i.e. the





**Figure 2.7** — The three molecular surface definitions. The Solvent Accessible Surface is shown in red, while the Solvent Excluded Surface is shown in purple. Both are defined by a probe, which is depicted in orange, rolling over the van-der-Waals surface of the molecule atoms, shown in blue.

space that the molecule occupies. This results in a major drawback: the van-der-Waals surface shows the general shape of the molecule but does not model the accessible region with respect to a solvent or other interacting molecules.

The Solvent Accessible Surface (SAS) [92] shows which parts of a molecule are accessible by another molecule, which is approximated by a probe of a certain radius, see central image in Figure 2.6. It is defined by a probe of a certain radius  $r$  that is rolled over the van-der-Waals surface without intersecting any atoms, see Figure 2.7. The centre of the sphere traces the SAS while rolling, therefore the SAS can be determined by increasing the radius of the van-der-Waals spheres by  $r$ . Since the probe represents a possible interaction partner, all atoms that are accessible to this probe contribute to the SAS. The SAS is therefore a feasible model for analysing possible binding partners. However, the inflated depiction of the molecule can lead to intersections with other molecules.

The Solvent Excluded Surface (SES) [125, 29] is also traced out by a probe rolling over the van-der-Waals surfaces of the atoms. However, the SES is determined by the contact of the the van-der-Waals surfaces and the surface of the probe, see Figure 2.7. The SES can be described analytically by using three basic geometric primitives [29]: convex spherical patches, toroidal patches and concave spherical patches. When the probe is in contact with only one sphere and it can move around freely on its surface, convex spherical patches are generated. When the probe is in contact with two spheres simultaneously it can only move along a circular arc. This generates toroidal patches. Concave spherical patches are generated when the probe is in contact with three spheres as it cannot move. The SES only shows atoms that are accessible by a probe sized atom or molecule, which gives a good impression of the molecular volume, since it does not enlarge the van-der-Waals spheres of the atoms, see right image in Figure 2.6. It can be classified into two parts, the contact surface and the reentrant surface. While the contact surface is made up of all atom surfaces that can be touched by the probe while rolling over the van-der-Waals surface, the remaining parts of the SES are the reentrant surface, i.e. the gaps between the atoms that are closed by the probe. This leads to a smooth representation of the molecular surface since smaller gaps between the atoms

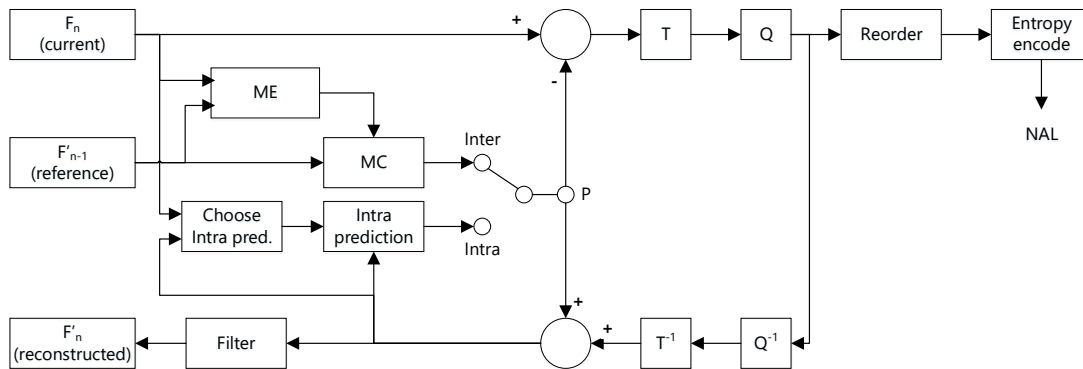
are closed while larger openings, such as channels, are still visible. Therefore the SES is ideally suited for the analysis of protein-solvent interaction or docking at binding sites.

In the past, numerous algorithms to compute the SES were developed [84]. Those algorithms can be split into ones that calculate an explicit mesh of the SES, such as MSMS by Sanner et al. [136], or ones that render the SES directly without calculating an explicit representation beforehand (e.g., Krone et al. [87] or Rau et al. [122]). The explicit mesh of the SES is created by either extracting the aforementioned geometric primitives or extracting the isosurface of the molecular volume from a discretised grid. The second extraction approach was used, for example, by Can et al. [26], Yu et al. [175] and more recently by Hermosilla et al. [60]. For the first extraction approach several methods were introduced to speed up the computation of the formulas that describe the geometric primitives. Sanner [135] developed the Reduced Surface algorithm, while Totrov and Abagyan [158] proposed the contour-buildup algorithm that uses the SAS to compute the SES. The definition of the Reduced Surface algorithm is similar to the aforementioned definition of the SES. If the probe is in a fixed position, i.e. in contact with three atoms, the centre points of the atoms form a polygon, the Reduced Surface face. Since all polygons can be subdivided into triangles the resulting Reduced Surface faces are triangles. For every Reduced Surface face a concave spherical patch is generated. The edges of the face indicate toroidal patches, while the vertices of the face represent the convex spherical patches. A more detailed description of the Reduced Surface algorithm can be found in [136].

The contour-buildup algorithm is also used by Krone et al. [87] on the GPU and Rau et al. [122] on the CPU since it can be parallelised efficiently. In the contour-buildup algorithm the path of the probe centre is either an arc or a full circle. First the extended sphere for each atom is computed, which corresponds to the SAS. These spheres now might intersect each other and the intersection can be described by a small circle. If more than two spheres intersect each other, the small circles might also intersect each other. In this case they are partially cut into circle arcs, the union of which defines the contour of a set of intersecting spheres, from which the SES can be derived. Each arc describes the path the probe takes while it generates a toroidal patch, while the concave spherical patches are the spheres that represent the atoms of the van-der-Waals surface. Again a more detailed description of the algorithm can be found in [158].

## 2.4 Image Transport

While in a local setting, the availability of the previously mentioned high-speed network interconnects like Infini-Band allow the transmission of uncompressed frames. However, this technology is not available everywhere. Therefore, compression algorithms have to be used if the image is transported between two computers. There are several methods



**Figure 2.8** — The H.264 encoder design, as described by Richardson [126].

that compress a single image, such as PNG, JPEG, but these might not reduce the required bandwidth enough in order to allow an interactive transmission of images, since temporal redundancies are not taken into account. An interactive visualisation can be interpreted as a video stream and therefore video compression codecs, such as H.264 or H.265 that use spatial and temporal compression in order to reduce the required bandwidth as much as possible, can be used.

In order to send the compressed video streams multiple protocols can be used. An example would be the Real-Time Streaming Protocol (RTSP) [120, 144], a network protocol that was designed for multimedia streams, e.g. video and audio streams. Clients can use it to control media servers by using commands such as play, pause, record. This allows real-time control of the media streaming, i.e. video on demand, from a server to a client. Other examples would be the TCP/IP or UDP protocols that allow for a simple transmission without any additional commands.

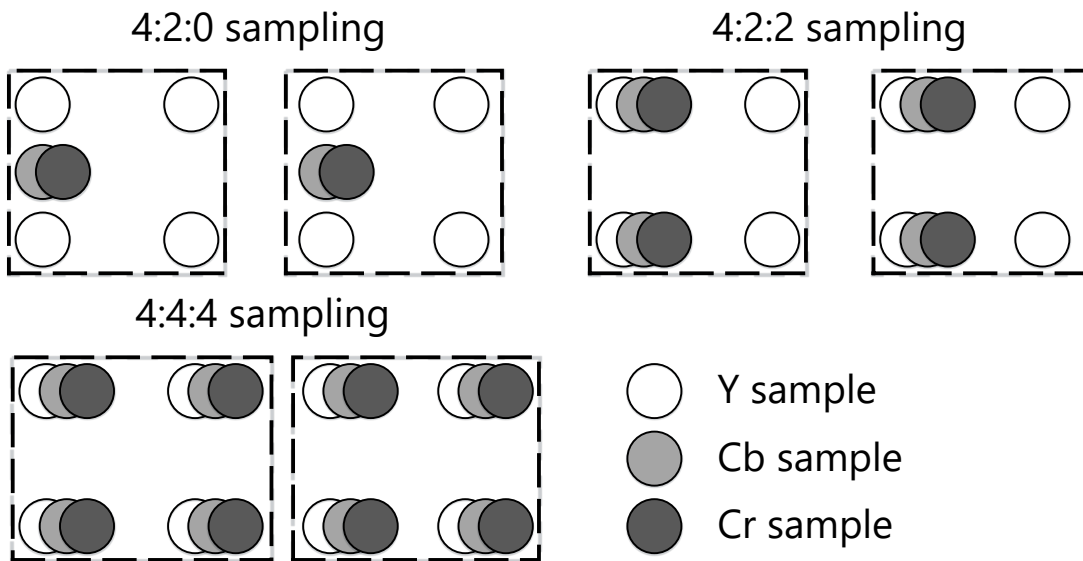
### 2.4.1 Encoder

The majority of the video coding standards released since the early 1990s are based on the same generic design, although there are many differences in detail between the standards and between implementations. This design consists of a motion estimation (*ME*) and compensation (*MC*) front end and a transform stage as well as an entropy encoder. For the H.264 codec, the intra prediction was added to the design, as shown in Figure 2.8. The encoder design has two paths, the first is the encoding path and the second is the reconstruction path. The encoding processes an input frame ( $F_n$ ) in macroblocks. Each macroblock contains a  $16 \times 16$  luma region and its corresponding chroma samples and is encoded in intra or inter mode based on the reconstructed picture samples in order to generate a prediction. In intra mode samples in the current slice

that have been previously encoded, decoded and reconstructed are used. In inter mode the motion-compensated prediction from one or two reference picture(s) is(are) used. In the next step a motion compensated prediction is computed and subtracted from the current macroblock to produce the residual (difference macroblock). Then the residual is transformed ( $T$ ) and quantised ( $Q$ ). In the final step the coefficients, motion vector and associated header information for each macroblock are entropy encoded to produce the compressed bitstream. In order to update the previously encoded frame each quantised macroblock is rescaled and inversely transformed to produce the decoded residual. The decoded residual is then added to the prediction in order to compute the reconstructed macroblocks of the reconstructed frame  $F'_n$ , which is used as the reference frame for the frame  $F_{n+1}$ .

### 2.4.2 Input Colour Format

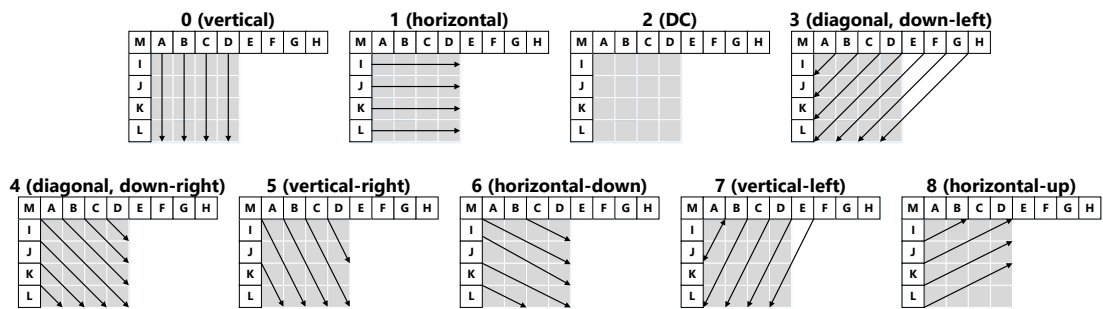
For rendering, the RGB or RGBA colour formats are used to represent images, with values in the range of  $[0, 255]$  for each colour channel. They therefore use 24bit (RGB) and 32bit (RGBA) per pixel. Additionally, the RGB(A) colour format does not weigh brightness and colours differently. Since the human visual system (HVS) is more sensitive to brightness than colours, choosing a different colour format makes more sense for compression, as the number of bits used for colours can be reduced. The YCbCr or YUV colour format separates the luminance (brightness) and chrominance (colours) into Y (luma), U (blue chroma) and V (red chroma) and can be converted to RGB(A) and vice versa. With this format, one can make use of the aforementioned fact that the HVS is more sensitive to luminance and use subsampling for the two chrominance channels. Chroma subsampling reduces the resolution of the two chrominance channels U and V in order to reduce the size of a single image. The subsampling is computed for a  $4 \times 4$  block of luma pixels and the resolution reduction for U and V can be varied. Possible subsampling includes 4:4:4, which uses no subsampling, 4:2:2 or 4:2:0 (cf. Figure 2.9) and can be expressed as a three part ratio -  $a:x:y$  which defines the chroma resolution in relation to a  $a \times 2$  block of luma pixels. With  $a$  being the horizontal sampling reference, which is usually 4,  $x$  being the number of chroma samples in the first row of pixel, i.e. the horizontal resolution in relation to  $a$ . And  $y$  is the number of changes of chroma samples between the first and second rows of a pixels. Therefore the first step of the encoding is to convert from RGB(A) to a YUV colour format, which can be done quickly by using a GPGPU based colour conversion. One of the most common YUV formats, used by video encoders, is NV12, which is a YUV 4:2:0 colour format where the U and V values are interleaved. Compared to RGB it uses half the bits per pixel, namely 12. Although Y, U and V all use 8 bit per value, there is only one U and one V value every four pixels, leading to the 12 bit per pixel.



**Figure 2.9** – Three possible configurations for the chroma subsampling of the YCbCr or YUV colour format. The 4:4:4 sampling uses no subsampling, while the 4:2:2 reduces the Cb and Cr samples by two, i.e. only every second pixel carries a U and V sample. The 4:2:0 reduces the Cb and Cr samples by four, i.e. one Cb and Cr sample per block of four pixels.

### 2.4.3 Intra Prediction

For spatial redundancy, i.e. regions of similar colour values, intra prediction is used. The intra prediction works with a block partition of the frame and predicts the colour values of a block using different algorithms. All possible predictions of the H.264 codec for a  $4 \times 4$  block are shown in Figure 2.10. The mode 0 (vertical), 1 (horizontal), 2 (DC) can also be used to predict the values of the entire  $16 \times 16$  luma component of the macroblock. An additional mode 3 (plane) is also available for the  $16 \times 16$  block, which fits a linear function to the upper and left-hand samples. The prediction is computed for all nine, in case of a  $4 \times 4$  block, or for all four, in case of a  $16 \times 16$  block, modes. In addition to the prediction the rate-distortion is calculated for all possible modes and the best one, i.e. the one with the lowest rate-distortion, is chosen for the final prediction. This final prediction is then subtracted from the current block in order to compute the residual, which is then encoded and transmitted to the decoder.



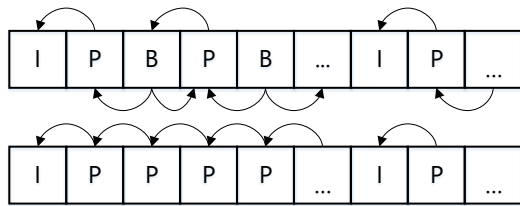
**Figure 2.10** — The nine (0 to 8) available prediction modes of the H.264 codec used for the intra prediction in a  $4 \times 4$  block, as described by Richardson [126]. The DC mode replaces in the current  $4 \times 4$  block by the mean value of the neighbouring pixel.

#### 2.4.4 Inter Prediction

For the temporal redundancy, i.e. regions of similar colour values for consecutive frames, inter prediction is used. The current frame is again divided up into blocks similar to the intra prediction, and the inter prediction finds the best matching block in the referenced frame, i.e. it tries to estimate the motion between the two frames. In order to find the best match, a search area in the reference is defined and the algorithm (motion estimation) compares all regions of the same size inside that area to the current block. The motion estimation looks for the lowest energy of the residual, i.e. the difference between the current block and the candidate region. It is therefore essential to speed up this algorithm by fast search methods, approximations or early terminations in order to find the best match as quickly as possible. For the best candidate the residual is computed as well as the motion vector, which encodes the offset between the current block and the position of the candidate region. Both the residual and the motion vector will be encoded and transmitted to the decoder.

#### 2.4.5 Frame Types

Video encoding commonly uses three different types of frames. The first is the I frame, also known as intra or keyframe, and is self contained, i.e. it does not depend on another frame and can be rendered directly after decoding, as it only uses intra prediction. It is also the first frame of a video stream as it is needed to start decoding. The second type is the P frame, or predicted frame, which makes use of the fact that the current frame can be rendered using information from the previous frame. It therefore only contains the difference between the previous and the current frame and uses either intra or inter prediction for each macroblock. The encoder can choose between intra and inter mode



**Figure 2.11** — Possible sequences for the encoder. Both start with an I frame, followed by a sequence of P or B frames either until the stream ends or another I frame is inserted. The bottom sequence only uses I and P frames in order to reduce the latency, while other sequence inserts B frames in order to reduce the size of the bitstream.

for each macroblock as it may be more efficient to encode a macroblock without motion compensation (in intra mode). The third frame type is the B frame, or bi-predictive frame, which uses the previous and the next frame to encode the current frame. Again it uses either intra or inter prediction for the macroblocks. This increases the encoding latency but can potentially reduce the required bandwidth as the information encoded in the B frames might be smaller than in P frames due to the fact that a past and a future frame is used. The three frame types are combined to define a group of pictures (GOP) that is used by the encoder, see Figure 2.11. The length of a GOP is the distance between two I frames. For an optimal low latency encoding the sequence would be I,P,P,... until the end of the stream, i.e. use an infinite GOP length. This, however, is difficult to maintain, because if a single P frame is lost the decoding introduces artefacts from which it never recovers. Therefore, some error resiliency method is required for noisy mediums, which either transmits I frames periodically or upon a request by the client.

### 2.4.6 Transformation, Quantisation and Entropy Coding

After the residual, either from intra or inter prediction, for the macroblock is computed it is transformed in order to further discard data while keeping the overall quality. For example H.264 uses either a Hadamard transform or a discrete cosine transform (DCT) based transformation for the residual data of the macroblocks. Both transformations are related to the Fourier transform and convert the macroblocks into blocks of the same size that contain frequency coefficients, making it easy to eliminate spatial redundancy. For images, most of the data is generally concentrated in the lower frequencies. Therefore, after the transformation into the frequency components, the coefficients of the higher frequencies can be discarded, reducing the amount of data needed to describe the image without sacrificing too much image quality. The decoder uses the inverse transformation, i.e. from blocks of frequency coefficients to macroblocks of the same size containing pixel values.

The next step in the encoder pipeline is the quantisation, which maps a signal with a range of values to a quantised signal with a reduced range of values, i.e. this is a lossy



operation. This is achieved by the simple formula:  $Z_{ij} = \lfloor \frac{Y_{ij}}{QStep} \rfloor$ , with  $Y_{ij}$  being the coefficient of the previous transformation,  $QStep$  the step size of the quantisation and  $Z_{ij}$  the quantised value. For example H.264 supports 52 values for the  $QStep$ , which are indexed by the Quantisation Parameter (QP). The QP can take values between 0 and 51 and the  $QStep$  double in size for every increment of six in QP. This range allows control of the trade-off between bitrate and quality, e.g. choosing a low QP value increases the bitrate and the quality, while a high QP value decreases the bitrate as well as the quality. As with the transformation the decoder uses the inverse operation:  $Y_{ij} = Z_{ij} * QStep$  to dequantise the received data. The methods described in section 5.1 and section 6.2 change the QP value for each macroblock to reduce the bitrate while keeping the image quality as high as possible.

The last step of the encoder is the entropy coding of the quantised data. This additional, lossless compression, further reduces the size of the data. There are multiple options, for example, H.264 uses either variable-length codes (VLCs) or context-adaptive arithmetic coding (CABAC) depending on the entropy encoding mode. The data of a residual block can also be encoded using a context-adaptive variable length coding (CAVLC) scheme. Entropy coding is used, among others, for the parameter of the quantiser, the motion vectors and the residual data.

The decoder needs to know about the decisions taken by the encoder, such as bit depth, resolution, predictions info (motion vectors, intra prediction direction), frame rate, frame type, frame number and many more. This information is sent to the decoder using a network-friendly structure, called Network Abstraction Layer (NAL). The NAL consists of multiple units separated by a synchronization marker. Usually, the first NAL of a bitstream is a sequence parameter set (SPS), which contains the general encoding variables, such as the resolution. Following the SPS is the picture parameter set (PPS), which contains parameters that apply to the decoding of one or more individual pictures inside a coded video sequence.

### 2.4.7 Decoder

The decoder (cf. Figure 2.12) receives a compressed bitstream, to which entropy decoding is applied, resulting in quantised coefficients. These are then scaled and inverse transformed to compute the residual block, which is identical to the residual block of the reconstruction path of the encoder. Based on the decoded header information of the bitstream, the decoder creates a prediction block, again identical to the prediction block of the encoder, to which the residual block is added, resulting in the final decoded block ( $F'_n$ ).



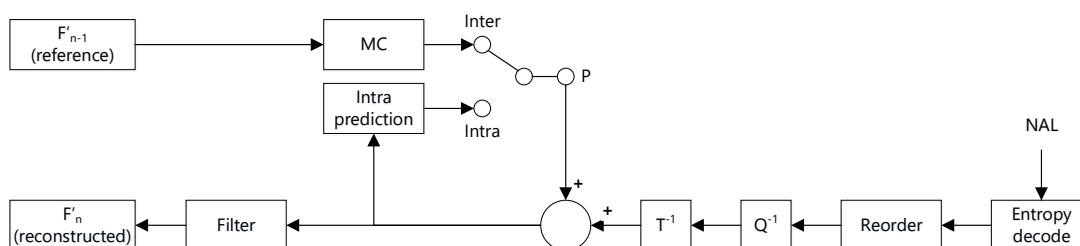


Figure 2.12 — The H.264 decoder design, as described by Richardson [126].

## 2.5 Software Infrastructure

The methods explained in chapter 3 rely on certain preprocessing steps for the input data and a software environment for the implementation. In this section the format of the input data as well as the necessary preprocessing steps, which are not part of the contribution of this thesis, are explained. Additionally, the MegaMol visualisation framework is introduced, which was used as a basis for the implementations discussed in chapter 3.

### 2.5.1 Input Data

The input data is the RCSB Protein Data Bank (PDB) [14] file format, which is one of the most widely used file formats for protein data. The RCSB PDB collects all known protein structures and assigns a unique 4-character identifier, the PDB ID, to each data set. Each data set contains a single line of information for each entity, for example, an atom of a protein. Each line starts with a keyword followed by the attributes of the entity, which are defined by a data type and the position, i.e. the column numbers. For atoms, the file format provides the element, a name, the position in  $\mathbb{R}^3$  and the residue that the atom belongs to. Additionally, there are optional attributes like the charge or temperature factor of an atom. Because the PDB file format is designed for static data, time-dependent data stored as trajectories in the GROMACS XTC [52] file format can also be used as input.

The preprocessing includes computing the Solvent Excluded Surface (SES) [125], which is one of the most useful and, therefore, most commonly used molecular surface representations. The SES is the interface of a molecule with respect to a spherical probe, i.e. the probe rolls over the union of spheres that represent the atoms of a molecule. The radius of this probe approximates a solvent molecule of a certain size. That is, small gaps between atoms that are not reachable by the probe are closed. For the methods described in chapter 3, the triangulated SES, as described in subsection 2.3.2, is used as

the input. The radius of the probe, typically a value between 1.5 and 3.0 Å, as well as the tessellation level for the triangulation of the molecular surface can be freely chosen by the user.

## 2.5.2 The MegaMol Framework

Parts of this subsection have been published in:

- P. Gralka, M. Becher, M. Braun, F. Frieß, C. Müller, T. Rau, K. Schatz, C. Schulz, M. Krone, G. Reina, T. Ertl. MegaMol – a comprehensive prototyping framework for visualizations. *The European Physical Journal (Special Topics)*, 227: Particle Methods in Natural Science and Engineering(14):1817–1829, 2019. [50]

MegaMol is a low-overhead prototyping framework for interactive visualisation of large scientific data sets. It originated from a joint research project<sup>2</sup> between biologists, physicists, material scientists and visualisation experts working with large, particle-based data sets that, for instance, come from molecular dynamics simulations. In the meantime, the software has evolved beyond that as new algorithms and techniques have been implemented to handle many diverse tasks. Furthermore, improvements on the software engineering side have been made, such as an easy-to-handle scripting interface. The programming model is organized in a modular scheme to be highly adaptable and uses modularization in order to re-use existing algorithms in different contexts. This also allows the creation of cutting-edge, and partially experimental, modules next to the stable core framework. MegaMol was implemented using C++ as the primary language, provides a Lua-based scripting interface and runs on Microsoft Windows and Linux. This cross-platform portability implies the use of OpenGL as the main graphics API.

MegaMol consists of the core library, a front end and optionally one or more plugins. While the core library manages the Module Graph, loads plugins and provides the low-level GPU access via the OpenGL graphics API, the front end provides the user interface. Each plugin provides one or more Modules that can communicate with each other and are placed in the Module Graph. This graph is acyclic and consists of at least one data source and one data sink, an example graph can be seen in Figure 2.13. Links between modules in the graph represent bi-directional data or information transport, which follows the pull paradigm. This was chosen because the primary results of MegaMol are interactive visualisations and therefore the data should traverse the graph only once a new image has to be rendered.

<sup>2</sup> SFB 716 “Dynamic Simulation of Systems with Large Particle Numbers”, <http://www.sfb716.uni-stuttgart.de/> (last accessed 22/04/2022)

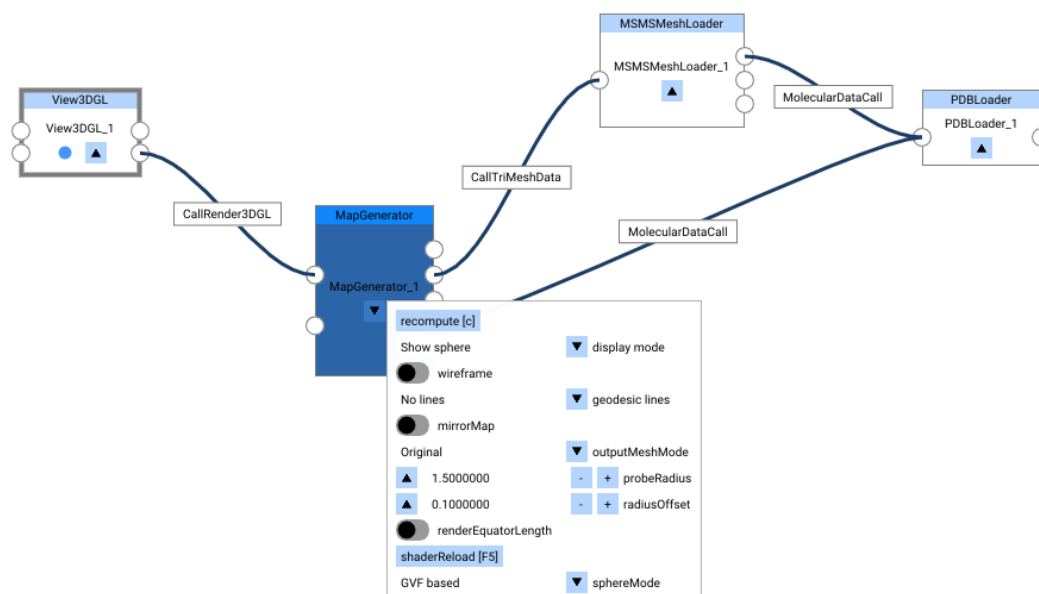
A Module represents a component in the functional logic of the data or visualisation processing pipeline, represented by the Module Graph. The scope of each Module is not enforced and therefore left to the developer. A Module can contain only a single algorithm or a complete rendering pipeline. For instance, the entire functionality of the deferred shading [56, 4] renderer can be implemented in a single Module, with the final image as output, or divided up into multiple Modules, each containing the functionality of a single rendering stage, and transferring intermediary buffers between them. The zero-copy paradigm in MegaMol dictates that every data entity produced by a Module is owned by this Module. This reduces the memory footprint and improves scalability.

The communication between the Modules is provided by Calls, which model intents. An intent specifies the reason for invoking a call, for example, to request new data, or inquire about the extents of the data. Again the granularity of a Call is not enforced and left up to the developer, although the recommendation is to bundle all intents of a Module in a single Call. Therefore, a single Call provides a complete interface to a specific data type, for instance, particle data. This also improves the re-usability of the Call. If the underlying data is reusable the Call is also reusable, i.e. all Modules processing particle data should implement the intents of the Call. Following the zero-copy paradigm, the Call transports only references of the data owned by a Module.

Additionally, caching and update checks help to avoid unnecessary computations. Therefore all Modules that provide or process data, compute a hash, i.e. a numerical value that only changes if the data changes, and provide it to the outgoing Call. This allows other Modules to check if there is new data and only then request it. For dynamic data sets, the combination of frame ID and data hash identifies the data and can be used to reduce unnecessary re-computations.

Every Module exposes Parameters that allow the user to interact with the functionality of the Module. A Parameter represents a value, including but not limited to a scalar, a colour, or a file path and also stores meta-information such as value bounds. It also allows the registration of a callback, which is triggered, once the corresponding value has changed. The Parameters are exposed to the Graphical User Interface (GUI) and are also accessible through the API of MegaMol. This allows the user to interact with them either through command line options, scripting, or through a GUI. The native GUI is built on ImGui [112] and is a minimalistic, straightforward front end that follows MegaMol's characteristic as prototyping framework. It exposes all Parameters of all Modules within the currently active Module Graph.

The Modules and Calls of the Module Graph are instantiated by the core library on request of the front end. Either the core loads this graph from an XML file or retrieves its content from the Configurator, see Figure 2.13. The Configurator analyses the core library as well as all available plugins and collects all Calls and Modules. It allows the user to graphically edit the Module Graph by adding or removing Modules, changing



**Figure 2.13** – The Configurator tool used to design Module Graphs. Each box with a light blue header represents a Module, while the lines between them are the Calls. The large box contains some of the Parameters of the MapGenerator Module used to create the Molecular Surface Maps described in section 3.1.

their Parameters, and connecting them with Calls. The final graph is then either saved as an XML file or forwarded to the core library. This also works while the Module Graph is active, i.e. the graph can be changed at run time.

MegaMol also supports mono and stereo cluster rendering on large high-resolution displays. For this, the execution on a GPU or CPU cluster is synchronised, i.e. all instances of MegaMol are synchronised and each instance only renders a part of the overall image based on a display topology provided by the user. The synchronisation and communication is performed via MPI, leveraging the high-bandwidth and low-latency networks usually available on such clusters. These instances are controlled by a single master instance that provides the initial Module Graph and transmits all parameter changes, including the camera state to keep the instances in sync.

## MOLECULAR SURFACES

### Parts of this chapter have been published in:

- M. Krone, F. Frieß, K. Scharnowski, G. Reina, S. Fademrecht, T. Kulschewski, J. Pleiss, and T. Ertl. Molecular Surface Maps. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):701–710, 2017. [86]
- K. Schatz, F. Frieß, M. Schäfer, P. C. F. Buchholz, J. Pleiss, T. Ertl, and M. Krone. Analyzing the similarity of protein domains by clustering Molecular Surface Maps. *Computers & Graphics*, 99:114–127, 2021. [140]

Chemically, the function of a protein is defined by the shape, as well as the physico-chemical properties, that is accessible to potential reaction partners. This interface is the molecular surface, which is generally considered to be influential for the function of a protein [84]. To visualise this interface of a protein, multiple definitions for molecular surfaces have been developed as outlined in section 2.3. One of the most commonly used representations is the Solvent Excluded Surface (SES) that shows the surface that is reachable by a smaller molecule such as a solvent or ligand, including cavities and channels. However, the SES is visually complex and suffers from occlusion, as do all three-dimensional depictions. In this chapter, algorithms to visualise and compare molecular surfaces are discussed. The focus of these methods lies on providing a visual summarisation of a molecule’s interface with its environment. This allows researchers to assess all molecular surface attributes at a glance and simplifies the analysis and comparison of different data sets or points in time. Additionally, they can be used to analyse time-dependent data from molecular simulations without the

need for animation. The presented methods consistently make use of the capabilities of modern, programmable GPUs to accelerate the computations as well as the rendering.

Section 3.1 describes a method that makes use of the world map metaphor to compute a two-dimensional representation of the SES that shows both the physico-chemical properties as well as the topography of the molecular surface in a single image. This resolves the typical problems that come with the three-dimensional representation of the SES, such as view-dependency and occlusion. The resulting two-dimensional representation, called Molecular Surface Map, provides an intuitive overview over the molecular surface and its properties.

Section 3.2 describes a method to hierarchically cluster the Molecular Surface Maps described in section 3.1, using an image-based similarity score. The idea behind the clustering is the commonly acknowledged fact in structural biology that visually similar surfaces also imply a similarity in the function of the underlying protein [24, 160]. Additionally, similar surfaces result in similar Molecular Surface Maps. The hierarchical clustering is presented as a dendrogram, which allows an interactive exploration and analysis of the results.

### 3.1 Molecular Surface Maps

Two-dimensional representations have been used for molecular structures, such as the sequence diagram. It shows a simple string of the amino acids that form the protein using a one letter code. Sequence diagrams can be enriched with additional information like secondary structures or information about binding sites [5]. Another two-dimensional visualisation for proteins, which is commonly used, is the Ramachandran plot [119]. The plot shows the backbone dihedral angles and is used to analyse and compare conformations and to estimate secondary structures. The Protein Data Bank [14] provides both, sequence diagrams and Ramachandran plots, for all recorded protein structures. While these two visualisations are useful to analyse the structure of a protein, they are not correlated to its interface. Self-Organizing Maps (SOMs) have also been proposed to create specific two-dimensional representations for molecular surfaces. Neurons arranged on a torus surface (a plane with periodic boundary conditions) have been used to produce a two-dimensional mapping [8]. A spherical SOM has been used as an alternative topology for lower distortion, while still being prone to occlusion[58]. Scharnowski et al. [138] used a deformable model approach to establish shape correspondence between two molecular surfaces for comparison. They applied a diffusion-based external force—the Gradient Vector Flow (GVF) introduced by Xu et al. [173]—for the morphing process to achieve a partial mapping of cavities or protrusions. Postarnakevich et al. [117] also used deformable models to project surface properties onto a sphere. A bijective mapping for triangulated molecular surfaces of genus zero

onto a sphere that uses a parametrisation based on spherical coordinates was developed by Rahi and Sharp [118]. Hass and Koehl [59] used a conformal mapping between triangulated molecular surfaces of genus zero and a sphere to measure the globularity of the molecule and to measure shape similarities of molecular surfaces. Schatz et al. [142] present a representation of molecular surfaces that focuses on binding sites, which are typically located within cavities. They show the cavity containing the binding site as well as the surface region directly surrounding the cavity entrance in a simplified manner that resembles a hat, where the brim depicts the surrounding surface region and the crown the cavity. Polyansky et al. [115] presented a method that creates 2D maps of helical dimers based on the distance and the rotation angle with respect to the helical axis. That is, their method cannot be applied to arbitrary molecules. Structuprints by Kontopoulos et al. [83] projects surface points onto a sphere along a straight line from the center of the molecule. Surface points can, therefore, overlap on the sphere—especially for surfaces of higher genus—while other parts can be devoid of surface points. While the resulting map creates a fingerprint of the molecule as intended by the authors, it does not give a complete overview of the interface.

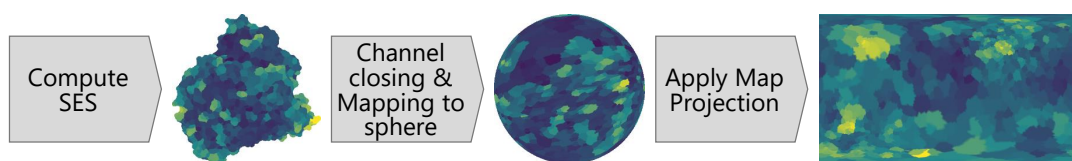
We introduce Molecular Surface Maps to overcome some of these defects. Our aim was to create a visualisation that shows the physico-chemical properties as well as the topography of the molecular surface in a single image, that summarises the surface and resolves common problems inherent to three-dimensional visualizations such as view-dependency and occlusion. To achieve this goal, we transferred the world map metaphor to molecular surface visualization. That is, like in cartography, the whole geometry of the molecular surface is mapped to a two-dimensional image. Thus, the analyst can get a quick overview of the whole surface and identify interesting surface features for further investigation.


### 3.1.1 Method

The algorithm to create the Molecular Surface Maps of a molecule consists of multiple steps. A short overview of these steps can be seen in Figure 3.1. In the first step the SES is computed, as described in subsection 2.5.1. The second step ensures that the surface is of genus zero, i.e. in this step any surface of genus  $n$ , with  $n$  greater than zero, is converted to genus zero. In the third step the molecular surface is mapped onto a sphere called the Molecular Surface Globe. In the fourth, and final, step the Molecular Surface Map is created from the Molecular Surface Globe by using a map projection.

**Molecular Surface** The first step, i.e. the computation of the underlying molecular surface, produces a triangulated surface mesh of the SES, since this is a viable choice for many different analytic tasks. In theory, it also works for other surface representations,



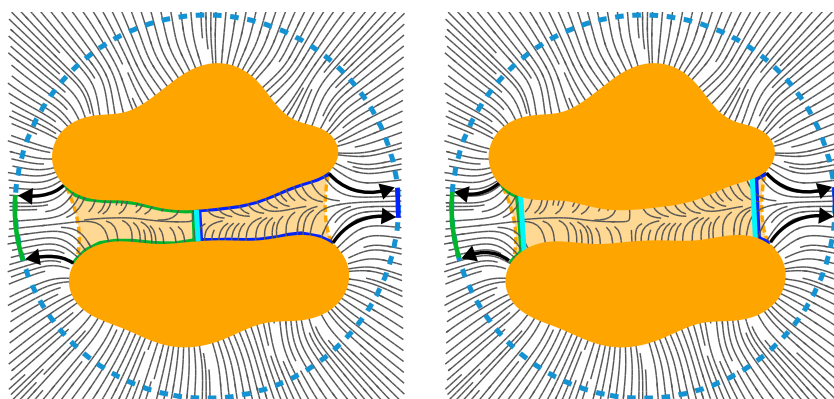


**Figure 3.1** — Schematic overview of the algorithm used to generate a Molecular Surface Map using the protein (PDB ID: 1AJN), which is coloured by temperature factor using the Viridis colour map (  ) from matplotlib [66]. The temperature factor is an indicator of the flexibility of the protein. The steps are depicted from left to right: computation of the SES, conversion from genus  $n$  to genus zero, mapping to the Molecular Surface Globe and the map projection that results in the Molecular Surface Map.

like Metaballs [16] or Skin Surface [36], as the following steps of the method only require a triangulated mesh.

**Channel Closing** While there are methods that map a molecular surface onto a sphere, they require a surface that is of genus zero, i.e. the mesh does not have any "holes". For example, a sphere has genus zero, while a torus has genus one. However, proteins in particular often have a surface of genus  $n$  due to channels or tunnels that run through the protein. Mapping a surface of genus  $n > 0$  to a sphere is not straightforward and the result is not intuitively interpretable, which contradicts the world map metaphor. Therefore, in the second step all channels need to be identified and their entrances and exits need to be closed in order to produce a surface of genus zero prior to the mapping to the sphere. A channel can be closed by placing a cut anywhere inside the channel and then closing the resulting holes in the mesh. The placement of the cut has an influence on the mapping onto the sphere, since it removes part of the channel, which is then no longer displayed on the sphere, as can be seen in Figure 3.2. Therefore the optimal cut would be at the halfway point of the channel, as both halves would be mapped onto the sphere. Longer channels would then either be mapped onto a very small area on the sphere or increase the distortion of the surrounding regions, making it harder to discern the information on the final map. This issue is also present in geographic cartography when mountains and valleys are flattened onto a globe, due to simply projecting them along the radial axis, resulting in the same problem: distances between sample points are not preserved and therefore the area will be distorted. Compared to molecular surfaces this is less of a problem, since the height of a mountain—or the depth of a valley, respectively—is very small compared to the radius of the earth. In order to solve this, an alternative solution can be used. Closing longer channels near

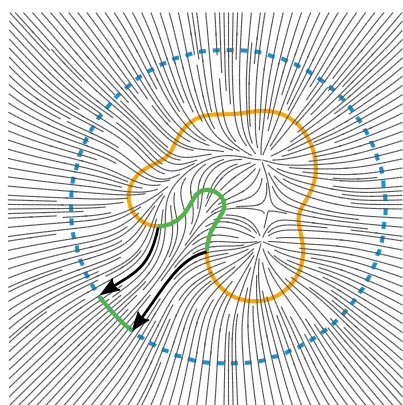




**Figure 3.2** – Two possible placements for the cut to close the channel. The cut on the left is in the middle of the channel and therefore the whole channel is mapped onto the sphere and visible in the final map. This, however, leads to a high distortion as both halves (green and blue) of the channel are mapped onto a small area on the sphere. The cuts on the right are placed close to the entrance of the channel, reducing the distortion of the mapping but losing information due to the fact that the interior of the channel is no longer mapped and not visible on the final map. © IEEE 2017 [86].

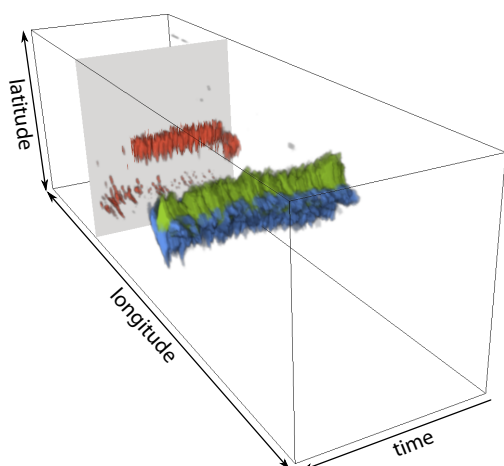
to the entrance lowers the distortion but also leads to a loss of information, since the interior part of the channel is no longer mapped to the surface and is therefore not visible on the final map. Both methods can be combined, i.e. closing longer channels at the entrance to reduce the distortion, while bisecting shorter channels to map them completely.

**Molecular Surface Globe** The third step in the algorithm uses the surface, which is of genus zero after the last step, to compute the Molecular Surface Globe by mapping the surface onto the sphere. This sphere is the bounding sphere of the molecular surface. Any of the mapping methods discussed in section 3.1 can be used to generate the sphere as all channels have been removed in the previous step. The mapping should be intuitively understandable by a human observer, therefore, the distortion should be as low as possible. Therefore, the location of an arbitrary point on the molecular surface with respect to other surface points should be similar on the globe. As a consequence of this, the areas on the molecular surface will correspond with the areas on the globe which makes the mapping intuitively understandable. Force-directed approaches, such as the one developed by Postarnakevich et al. [117], which morphs a tessellated sphere to the molecular surface, produce the required results. The method developed by Hass and Koehl [59] works similarly but in the opposite direction, i.e. they morph the molecular surface into the sphere. For the computation of the forces that pull the molecular surface towards the sphere, the sphere can be represented implicitly by its



**Figure 3.3** — The gradient vector flow (GVF) field that maps the molecular surface (orange) to the target sphere (blue, dashed). For clarity the distance between the surface and the sphere is exaggerated. Without internal forces, which try to keep the distance between points similar, the green cavity is mapped to a comparably small area on the sphere. © IEEE 2017 [86].

center and radius. This allows for higher accuracy, since an explicit target shape would require voxelisation and, hence, introduce additional discretisation errors. Another approach is to use a deformable model [156] to map a point on the molecular surface to a point on the sphere. Deformable models distinguish between external and internal forces which both act on every vertex of the surface, allowing for additional control over the ways the input surface is deformed. The modified GVF [138] can be used to obtain the external forces, which is illustrated in Figure 3.3. This algorithm creates a force field by diffusing both the individual components of the molecular surface normals (the source shape) and the sphere normals (the target shape). A major drawback of the aforementioned force-directed approach is the fact that the quality of the mapping depends on the input parameters, potentially requiring multiple tries until the result fulfils the requested quality standards. Rahi and Sharp [118] presented an alternative approach that does not use a force-directed morphing but a re-parametrisation of the vertices of the molecular surface. This is done by assigning spherical coordinates ( $\theta$  and  $\phi$ ) to every vertex in order to map them onto a sphere. As mentioned previously the complex shape of the molecular surface, which has protrusions and cavities, introduces distortion regardless of the chosen mapping algorithm. The morphing will generate a non-uniform distribution of points on the surface of the sphere. This relates to the issue mentioned above that also exists for geographical globes of the Earth: If a cavity is flattened onto the sphere, either the distances between the original sample points will be reduced or the points surrounding the cavity will be forced apart. The same is true for a protrusion. Since the goal is to create a meaningful and intuitive transformation of the surface mesh vertices onto the sphere, the original distances between the vertices of the surface mesh need to be maintained as much as possible. Therefore, both the force-directed approach as well as the parameter-based method will result in a pareto-optimal mapping in which points within dense regions try to relax without pushing the surrounding point further away. As a result, the area of a feature on the surface of the sphere will not exactly correspond to its original area on the molecular surface.



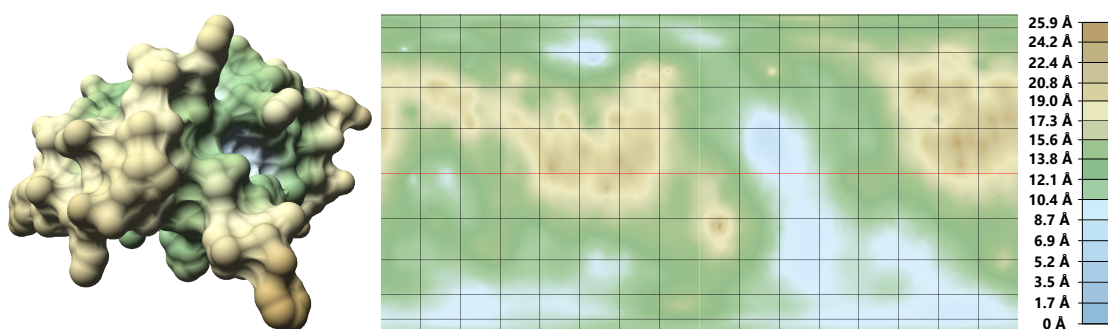
**Figure 3.4** — Space-time Cube showing the evolution two binding sites for a protein simulation. The uninteresting parts of the Molecular Surface Maps have been removed using the filtering operation and only the two binding sites are shown. Towards the end of the simulation the red coloured binding site vanishes because the protein has deformed so that it is not accessible any more. The grey clipping plane has been added as a depth cue. © IEEE 2017 [86].

**Map Projection** The final step of the algorithm produces the Molecular Surface Map based on the Molecular Surface Globe and a map projection. For this step any arbitrary map projection can be used to transform the sphere to a map. Snyder and Voxland [149] presented a comprehensive overview of map projections along with the respective equations. Although the mapping from the SES to the Molecular Surface Globe already introduces distortion and does not necessarily preserve the areas of the original SES, a cylindrical equal-area map projection such as Lambert can be a reasonable choice, as it does not introduce an even greater distortion during the projection. Several alternative versions of the Lambert projection have been proposed, such as Hobo-Dyer or Gall-Peters, and can alternatively be used in this step. A different option would be the Plate Carrée projection, i.e. a cylindrical equirectangular map projection<sup>1</sup>. This type of projection is equidistant along the meridians. Both types of projections create rectangular maps, which are not only commonly used in atlases of the world but also visually similar to the variant of the Mercator projection used by Google Maps, which makes it easier for users to relate to the resulting Molecular Surface Map.

### 3.1.2 Molecular Surface Map Colouring

As with geographical cartography, colouring is a vital part of the Molecular Surface Maps, since it shows the features of the projected molecule. Although the Molecular Surface Maps can be used with all kinds of molecules, the colourings focus on proteins, since the map representation was designed to be used in structural biology. All common (bio-)chemical colouring schemes can be applied to Molecular Surface Maps, since the

<sup>1</sup> Lambert equal area: [https://en.wikipedia.org/wiki/Lambert\\_azimuthal\\_equal-area\\_projection](https://en.wikipedia.org/wiki/Lambert_azimuthal_equal-area_projection), Hobo-Dyer: [https://en.wikipedia.org/wiki/Hobo%E2%80%93Dyer\\_projection](https://en.wikipedia.org/wiki/Hobo%E2%80%93Dyer_projection), Gall-Peters: [https://en.wikipedia.org/wiki/Gall%E2%80%93Peters\\_projection](https://en.wikipedia.org/wiki/Gall%E2%80%93Peters_projection) and Plate Carrée: [https://en.wikipedia.org/wiki/Equirectangular\\_projection](https://en.wikipedia.org/wiki/Equirectangular_projection) (last accessed 22/04/2022)



**Figure 3.5** — Cartography-inspired colouring by elevation. The colouring illustrates valleys and mountains of the original protein surface (left, PDB ID: 1RWE) on the Molecular Surface Map (right, Hobo-Dyer projection). © IEEE 2017 [86].

algorithm described above simply maps points on the molecular surface to the Molecular Surface Map. Colouring schemes that illustrate properties which influence the molecular interface are particularly useful. This includes hydrophobicity, temperature factor, or the electrostatic potential at the molecular surface. Since these properties are signified by a single value for each point on the molecular surface, a colour gradient is typically used for rendering. Categorizing colourings by physico-chemical properties of the amino acids is also useful.

Another important property is the availability of binding sites, which consist of a set of amino acids that have to be exposed to the molecular surface in order to be accessible. For visualisation, all atoms that belong to an amino acid of the same binding site are assigned the same colour (see Figure 3.4). A related colouring mode shows the binding of solvent molecules to the molecular surface. For each point on the molecular surface, the distance to the closest solvent of a specific type (e.g., water) is computed. If a solvent molecule is close to the surface, i.e. within 3 Å, it is classified as bound to the surface and the surface point is highlighted using colour. For a simulation, the binding can be aggregated over time, so the surface colouring can show preferred binding sites of this solvent [139].

Colour can also illustrate the topography (i.e., the shape) of the molecular surface. The geography-inspired colouring maps the elevation of a point to a colour (see Figure 3.5). While geographic maps usually use the sea level as reference point, for this colouring the elevation is determined as the distance to the center of mass of the molecule. An alternative is to use the length of the path that a point of the molecular surfaces has to travel until it reaches the Molecular Surface Globe. This colouring can also be used to identify deformations when comparing different conformations of the same molecule. Another option is to highlight topographical features like cavities or protrusions. For

example, available software like 3D-Surfer [91] can be used to extract all amino acids that form a cavity or protrusion, which can then be highlighted by colour on the Molecular Surface Map. Another possibility to highlight the shape is to compute the Ambient Occlusion (AO) [179] of the original molecular surface and show the AO factors on the map. The AO highlights topographical features like cavities or protrusions by making these areas darker compared to rest of the surface. Therefore, the shape of the surface is easier understandable on the Molecular Surface Map.

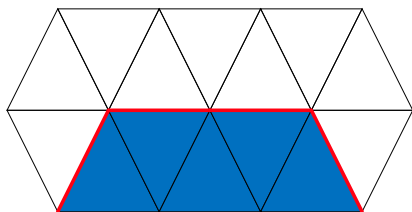
### 3.1.3 Extension to Time-dependent Data

Since Molecular Surface Maps are a two-dimensional representation of the molecular surface, they can be used to represent dynamic data sets using Space-time Cubes. Space-time Cubes are commonly used in geovisualisation to show the temporal evolution of data points or features on a map [55]. Therefore, a Molecular Surface Map needs to be computed and stored as a slice of a volume, for each time step of the simulation. This requires the molecule to be aligned for all time steps, which can be done as a preprocessing step by applying the commonly used Root Mean Square Deviation (RMSD) alignment. The review by Bach et al. [9] gives a taxonomy of commonly used operations. The usefulness of Space-time Cubes for Molecular Surface Maps depends largely on the map colouring, which in turn is determined by the analysis task. A common task in structural biology is to monitor the presence of pockets or the availability of binding sites throughout a molecular simulation. For this, the volume of the Space-time Cube can be visualised using the *oblique flattening* operation, that is, as a perspective projected cube. To make the development of surface features visible over time, uninteresting parts have to be removed using the *filtering* operation. In the case of binding sites or pockets, the neutral colour has to be filtered, that means that only the surface features will remain. Features can either be visualised using volume rendering [90] or by extracting isosurfaces. An isosurface extraction would implement the *aggregation* operation. The analyst can now see how a surface feature evolves and whether it is persistent in time or if it vanishes at some point. An example of this can be seen in Figure 3.4, where the red binding site vanishes at some point in time. To enable the full analysis of the data, camera adjustment has to be supported to prevent occlusion (*rotation* and *shifting* operations).

### 3.1.4 Implementation Details

The following paragraphs will describe the implementation of the steps of the algorithm, described in subsection 3.1.1. Since the preprocessing of the input data as well as the computation of the SES has already been described in subsection 2.5.1 it is not included here.





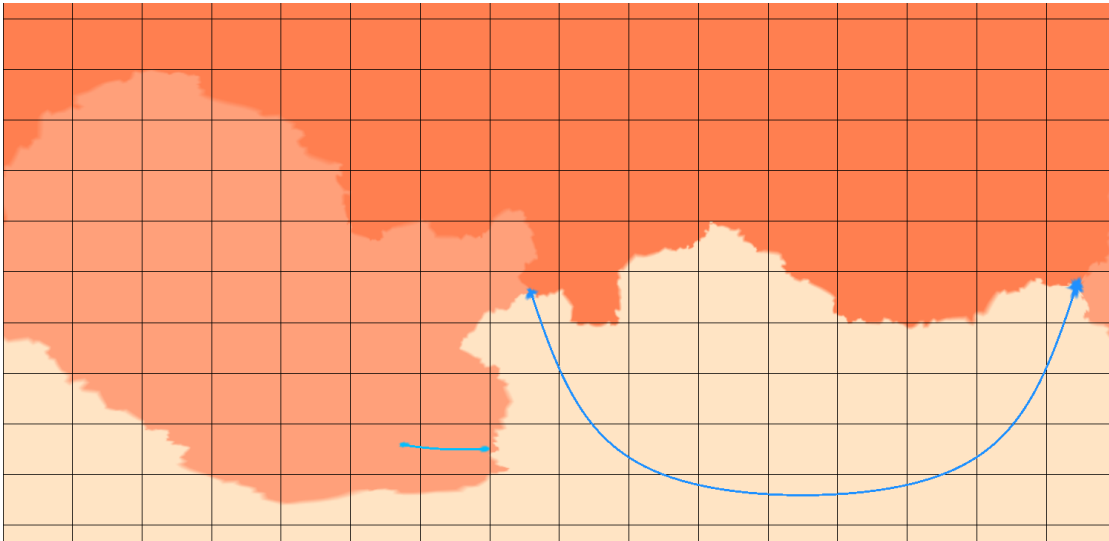
**Figure 3.6** — Part of a circle  $\Omega$  of faces (blue) that have at least a neighbouring face (white) that is not marked. The exact circle  $\omega$  is comprised of the vertices of the edges (red) between a marked face (blue) and an unmarked face (white). These vertices are used to compute the centre of the triangle fan that will be inserted to close the channel.

**Channel Closing** In the second step in the algorithm, described in subsection 3.1.1, the channels of the protein need to be found and closed. The first step is to check the genus of the protein by computing the Euler characteristic. If the molecular surface mesh is already of genus zero the remaining computations can be skipped. If it is not, then as previously discussed, the longer as well as the shorter channels need to be detected and closed. For this three methods are provided. The first uses Ambient Occlusion (AO) to detect longer channels and a Reeb graph to detect shorter channels, the second uses a Voronoi based method, while the third is an iterative process for computing the SES.

The AO method is an extended version of the method, presented by Krone et al. [89, 88]. The AO value for every vertex is computed in parallel on the GPU using 32 normal distributed rays. The number of rays that intersect the surface is counted and divided by the total number of rays, resulting in a value of 0 if all rays intersect the surface and a value of 1 if there are no intersections. After the initial computation of the AO values a smoothing function is applied. The AO value ( $AO_i$  of each vertex  $i$ ) is updated by the weighted average ( $AO'_i$ ) of all AO values from vertices inside a sphere with the radius  $d_{max}$ . First the distance  $d < d_{max}$  between two vertices ( $i$  and  $j$ ) is computed and then the AO value is added to the average, with a decreasing weight based on  $d$ .

$$AO'_i = \left( AO_i + \sum \min \left( 1, \frac{1}{d} \right) * AO_j \right) / C \quad (3.1)$$

$C$  is the overall number of vertices inside the sphere. Then, based on a defined threshold  $t \in [0, 1]$  vertices are marked as occluded if their AO value is below the threshold. Therefore, the threshold value has a direct influence on the detection of the channel. An adequate default value is 0.1, with this threshold most of the longer channels are detected completely. Each face is categorised based on the state of its vertices. If a face has three vertices that are marked as occluded the face is marked as occluded too. After this, the faces are grouped together in order to identify the faces that belong to the same channel or cavity, this is done by a parallel GPU based region growing algorithm. Once the faces are grouped each group needs to be identified as either a channel or a cavity. A group is part of a channel if it has more than one border with a group of unmarked faces,



**Figure 3.7** — Molecular Surface Map of a protein with two channels (PDB ID: 2BT9, map created using the parameter-based method and Plate Carrée projection). The colouring shows the three amino acid chains that form the protein. The entrances of the two channels are visible as blue areas on the map. The channel connectivity is illustrated using geodesic lines in the same colour as the entries that are connected by the original channels. © IEEE 2017 [86].

otherwise it is a cavity and can be ignored. For the remaining groups each entrance, i.e. the circle  $\Omega$  of faces that have at least a neighbouring face that is not marked, needs to be closed. First, the exact circle  $\omega$  is computed by iterating over the faces of the circle  $\Omega$  and identifying the edge that is shared with the unmarked neighbouring face, see Figure 3.6 for an illustration of this. All vertices of these edges are used to compute the centre of the triangle fan that will be inserted, by averaging their positions. This new vertex is then connected with the vertices of  $\omega$  and tessellated so that the new faces have roughly the same size as the existing faces. After all circles have been closed the vertices and faces of the group are removed from the mesh of the molecular surface. After the removal of the longer channels the mesh might still be of genus  $n > 0$ , therefore, the second part of the method uses a Reeb graph to detect the remaining channels or handle-like structures. For this the method of Dey et al. [33] is used, which finds minimal tunnel loops in a surface mesh using a Reeb graph. Using these loops the remaining channels are closed using the aforementioned cutting algorithm, i.e. the border circles  $\Omega$  are closed by inserting a triangle fan and the faces of the channel are removed.

The Voronoi method is based on the work of Lindow et al. [95], which was developed

to visualise potential molecular paths. For this they compute the Voronoi diagram of spheres from the van der Waals spheres of the atoms that make up the protein. A filtering process then selects the paths from the diagram that are of interest. We use the *Radius Filter*, which removes all edges along which a sphere of a certain radius cannot move without colliding with an atom of the protein. After applying the filter, we have a list of channels through the protein that can be closed using the aforementioned cutting algorithm, i.e. the border circles  $\Omega$  are closed by inserting a triangle fan and the faces of the channel are removed.

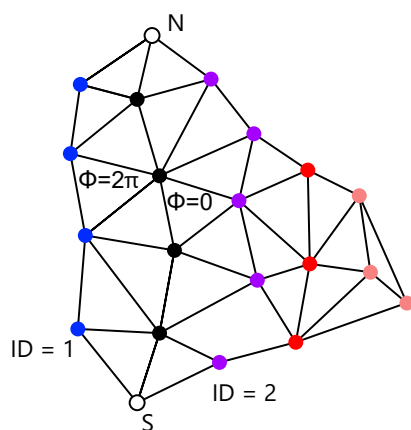
A usually faster method to close the channels is to iteratively compute the SES for different probe sizes until the resulting molecular surface mesh is of genus zero. As proposed by domain experts the radius of the probe should be between 2.4 Å and 4.0 Å for this method. Each iteration increases the radius of the probe by 0.2 Å until the resulting mesh is either of genus zero or the radius of the probe is larger than 4.0 Å. In this case the previously discussed more precise, but slower, approach to detect and close channels can be used.

As mentioned above, the closing of holes through the molecular surface (channels and handle-like structures) modifies the surface. The newly created triangles that close the holes are coloured individually to be clearly visible. All entries of a closed hole have the same colour, so that users can easily see which entries are connected. To further facilitate the visual analysis, entries of a hole can additionally be connected using geodesic lines on the globe and on the map. These lines resemble the widely used aeroplane route metaphor and clearly illustrate which entries are connected, see Figure 3.7.

**Molecular Surface Globe** The first step for computing the Molecular Surface Globe is to determine the bounding sphere of the SES mesh using the Bouncing Bubble algorithm [157]. This computes the approximate smallest enclosing sphere for a set of points. For the Molecular Surface Globe these points are the vertices of the SES mesh. Once the size of the bounding sphere is determined the morphing of the molecular surface to this sphere can begin. For this, two different algorithms can be used, the first employs the deformation method by Scharnowski et al. [138], which is based on the GVF and is a force-directed approach. The second is the parameter-based algorithm by Rahi and Sharp [118].

For the computation of the GVF, used by the first algorithm, not only the gradient of the target shape but also the gradient of the input shape is used. Therefore, the normals of the SES mesh and the ones of the sphere are sampled to a three-dimensional grid with grid spacing  $\Delta_g$ . A default value of  $\Delta_g = 1.0$  Å ensures subatomic detail. This grid is the input for the iterative diffusion process of the GVF, which generates the external force field for the following mesh morphing step. Depending on the shape and spatial extent of the input data, the number of iterations until convergence differs



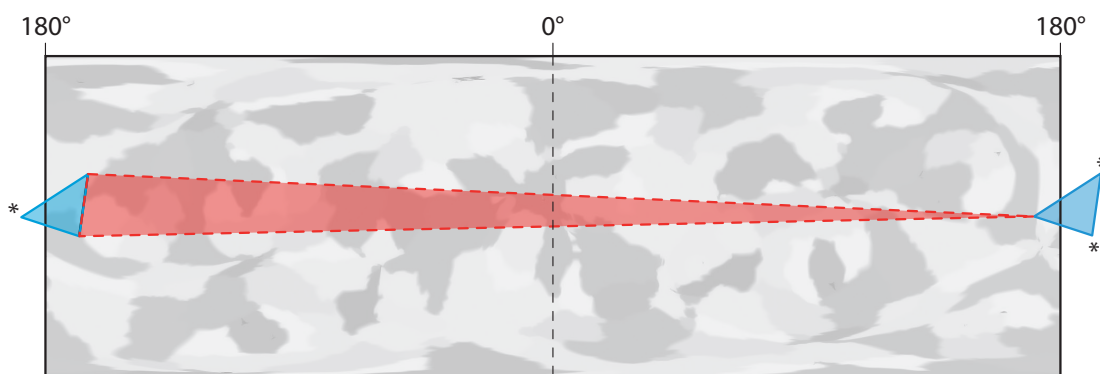


**Figure 3.8** — Assignments of the  $\phi$  values for the parameter-based algorithm as described by Rahi and Sharp [118]. For all blue vertices the  $ID = 1$  is used and for all purple vertices the  $ID = 2$ . The boundary meridian (black) connects the north (N) and south pole (S). During the propagation, using the Gauss-Seidel relaxation, the remaining vertices are assigned the unweighted average  $\phi$  value of their neighbours.

highly. The internal forces, on the other hand, require only the computation of a mesh Laplacian. This can be achieved with the summation described by Shen et al. [148]. The computations for both the external forces and the internal forces were implemented in CUDA and run in parallel on the GPU.

Both, the internal and external forces, are used to morph the molecular surface mesh onto the sphere. That is, the vertex positions are iteratively moved according to the combined forces until convergence. For an appropriate step length, the forces are normalized and scaled by the grid spacing  $\Delta_g$ . Additionally, the external forces are rescaled by  $-0.5$  every time the spherical target surface is crossed, which leads to quicker convergence and higher accuracy of the final vertex positions. The morphing can be influenced with the weighting parameter  $\mu \in [0 \dots 1]$ , which defines the ratio between external force and internal force. That is, external forces are scaled by  $\mu$  and internal forces by  $1.0 - \mu$ . A higher value of  $\mu$  leads to a less regular mesh, while a lower value of  $\mu$  favours the regularising internal forces. On the other hand, a lower value of  $\mu$  increases the distortion of the mapping relation. The default value for  $\mu$  is  $0.5$ , so neither the internal nor the external forces are favoured. The overall iteration process ends if the average displacement of all vertices is smaller than a minimum displacement  $d$ . The standard value of  $d$  was chosen to be  $0.001 \text{ \AA}$  as this yields good results. The final mesh is defined by the converged vertex positions while maintaining the triangle connectivity of the source mesh. The morphing process is also parallelised via CUDA.

For the parameter-based algorithm the first step is to identify two vertices that will become the north and south pole respectively (see Figure 3.8). In the original paper the authors used the two vertices with the largest Euclidean distance. This would, however, change the orientation of the Molecular Surface Globe compared to the original orientation of the molecule. Therefore the north-south axis of the molecule is determined, based on the up-vector of the camera, and the two vertices that are closest to this axis become the north and south pole respectively. Neighbours of the north pole



**Figure 3.9** – The red triangle crosses the antimeridian at  $180^\circ$  on the globe and stretches across the entire Molecular Surface Map after the map projection. Therefore a correction step is necessary, since these triangles cause z-fighting and jagged edges. In the geometry shader these triangles are duplicated and the two blue triangles with corrected vertex positions (\*) are emitted. This results in an artefact free rendering since the blue triangles are automatically clipped at the map boundary by the view frustum. © IEEE 2017 [86].

are then assigned the value  $z = +Z$  and neighbours of the south pole are assigned the value  $z = -Z$ , with  $Z > 0$ . Every other vertex is initialized with the value  $z = 0$ . Using the Gauss-Seidel relaxation they are updated using the unweighted average of their neighbours. After convergence, the  $\theta$  values are computed using the inverse Mercator projection, with the  $z$  values as input. The boundary meridian can be detected by starting at a random neighbour of the north pole and following the path of steepest descent until a neighbour of the south pole is reached. The vertices of the meridian are assigned the  $ID = 0$ . Vertices on the left side are assigned the  $ID = 1$  and vertices on the right side are assigned the  $ID = 2$ . Then the  $\phi$  value of all non-meridian vertices is set to zero. The  $\phi$  values are again propagated using the Gauss-Seidel relaxation using the unweighted average of their neighbours. If the meridian is approached by a vertex of with  $ID = 2$ , a  $\phi$  value of 0 is propagated, if the meridian is approached by a vertex with  $ID = 1$ , a  $\phi$  value of  $2\pi$  is propagated.

**Map Projection** Map projections are implemented using the OpenGL render pipeline. The vertex shader projects the triangles into the two-dimensional domain by transforming the vertices using either the equations for the cylindrical equal-area projection

(see equation 3.2) or the equirectangular one (see equation 3.3):

$$\text{Lambert:} \quad x = \frac{\text{atan2}(v_x, v_z)}{\pi} \quad y = v_y \quad (3.2)$$

$$\text{Plate Carrée:} \quad x = \frac{\text{atan2}(v_x, v_z)}{\pi} \quad y = \frac{2 \cdot \text{asin}(v_y)}{\pi} \quad (3.3)$$

where  $(v_x, v_y, v_z)$  is the normalized vector from the bounding sphere center to the triangle vertex. The values are scaled to  $[-1, 1]$ , since the result is rendered into an off-screen fragment buffer with the appropriate pixel ratio for the respective map projection. In the geometry shader, triangles that cross the antimeridian, which is at  $180^\circ$ , will be corrected. These triangles stretch across the entire map, resulting in a defective rendering that is visible through z-fighting and jagged edges around the map. Therefore, these triangles are duplicated, as illustrated in Figure 3.9. The first triangle all vertices that cross the meridian are translated by  $360^\circ$  and for the second triangle all vertices that do not cross the meridian are translated by  $-360^\circ$ . The resulting triangles will be clipped automatically at the map boundary by the view frustum.

### 3.1.5 Results and Discussion

The performance of the algorithm was tested using protein data sets from the Protein Data Bank [14]. Table 3.1 shows the computation times (excluding time for SES computation via MSMS [136]). All tests were run on a system equipped with an NVIDIA Quadro M6000, two Xeon E5-2640v3 CPUs and 256 GB of RAM. Both, the force-directed and the parameter-based method, were implemented using CUDA. As can be seen, the force-directed is faster than the parameter-based approach, this is mainly due to the time it takes the two Gauss-Seidel relaxations to converge.

Additionally the iterative approach is usually faster than the other two channel closing methods except for the 3ZHB protein where it requires multiple iterations to close all channels. Otherwise it only required one iteration to generate a mesh of genus 0 and was therefore significantly faster than the other methods, since computing the SES takes roughly the same time. Also the computation of the MSM is usually faster since the mesh has fewer vertices. The other two channel closing methods produce similar cuts, but the Voronoi based approach is about four times faster than the AO and Reeb graph method.

As anticipated, the force-directed approach often creates a slightly more intuitive mapping of the SES for globular molecules. While the parameter-based method also produces very good results, it can lead to a less intuitive mapping of SES regions due to the relaxation of the  $\phi$  and  $\theta$  values. However, the difficulty of finding a good combination of parameters for the force-directed approach increases with the complexity of the underlying molecule. Especially for proteins with deep cavities, the method is very

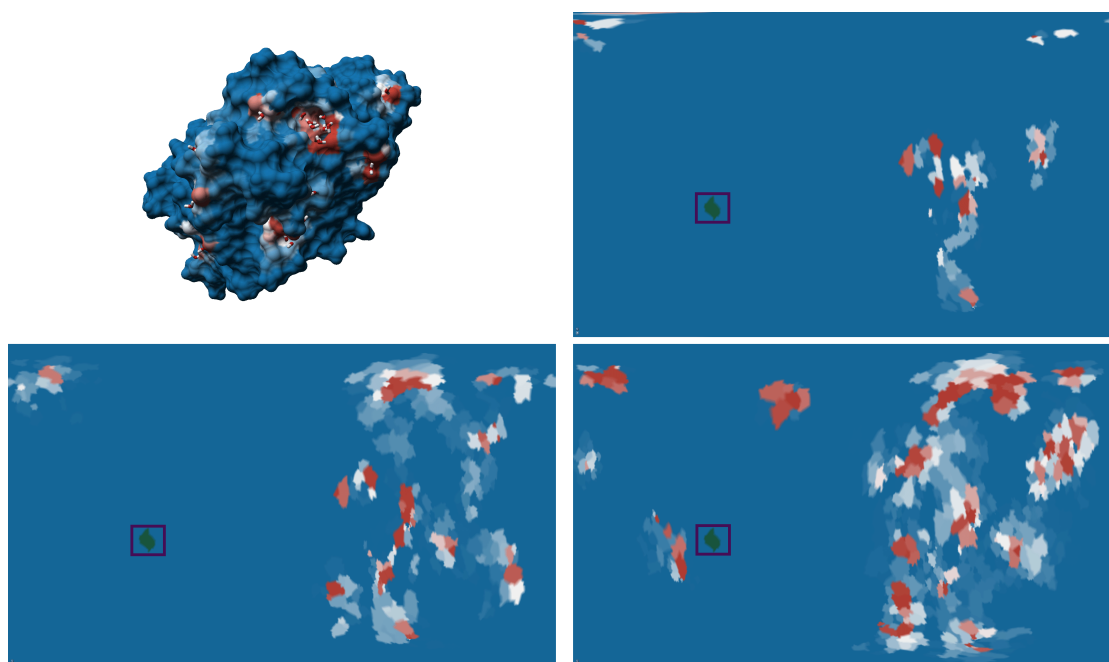
**Table 3.1** — Comparison of the computation times of the force-directed method (FD) and the parameter-based method (PB) for four proteins of different sizes and different genus, denoted by  $G$  SES. Timings are denoted as follow and all are measured in seconds: *AO*: the AO based method based on [89] and [33]; *VB*: the Voronoi based method based on [95]; *IT*: the iterative channel closing; *MSG*: Molecular Surface Globe computation; The *AO*, *VB* and *IT* measurements denote the time it takes to compute the individual channel closing method on the input SES.

	#atoms	G SES	AO	VB	IT	MSG
<b>FD: 1RWE</b>	816	0	–	–	–	0.24
<b>PB: 1RWE</b>	816	0	–	–	–	0.63
<b>FD: 1TCA</b>	2324	0	–	–	–	0.30
<b>PB: 1TCA</b>	2324	0	–	–	–	1.51
<b>FD: 3ZHB</b>	4016	4	8.23	2.79	11.18	0.68
<b>PB: 3ZHB</b>	4016	4	8.38	2.75	11.15	3.84
<b>FD: 1AF6</b>	10,050	8	36.79	7.57	2.77	0.89
<b>PB: 1AF6</b>	10,050	8	31.25	7.74	2.67	11.96

sensitive to changes in the ratio between external forces and internal forces. On the one hand, if the internal forces are weighted too high for proteins with deep cavities, the algorithm might not converge. On the other hand, if the internal forces are weighted too low, the triangles within the cavity will be mapped to a very small area, leading to high distortion. In contrast, the parameter-based method is able to produce a valid mapping even for very complex cases and requires no user-defined parameters.

As mentioned in subsection 3.1.1, mapping the SES mesh to the Molecular Surface Globe will result in a non-uniform distribution of triangles. Triangles that lie on protrusions or cavities of the SES will lead to denser regions on the sphere: The triangle area will be lower, which results in a locally higher vertex density. Due to the corrugated nature of the SES, this issue occurs for all geometric mappings that preserve the triangle connectivity. For both the force-directed method and the parameter-based one, the area of the smallest triangle and the average triangle area are approximately the same. The parameter-based method creates on average more smaller triangles than the force-directed one. As a result of this analysis it becomes clear that the parameter-based approach is better suited for more complex surfaces, while the force-directed approach is better suited for more globular molecules without deep cavities or large protrusions.

Zaks and Klibanov [176] showed that a small amount of water is necessary for protein activity. Therefore, water can be considered a lubricant that is necessary for enzyme activity. Figure 3.10 shows several molecular dynamics simulations of *Candida an-*



**Figure 3.10** – Top left: Solvent Excluded Surface of *Candida antarctica* lipase B (CALB) with water molecules bound to the protein. The surface is coloured according to the accumulated water binding over the whole simulation time (blue: low water affinity; red: high water affinity). The three maps show different simulations with increasing water activity, that is, more water molecules (from top right to bottom right: 6, 25, and 50 water molecules). The dark green spot, highlighted by the purple rectangle, shows the location of the ligand binding site. The maps were generated by the force-directed method and Gall-Peters projection. © IEEE 2017 [86].

*tarctica* lipase B (CALB) at various water activities that were performed using the GROMACS [162] simulation package and the OPLS force field. Using the Molecular Surface map of the aggregated solvent binding for the whole trajectory revealed that water is not homogeneously distributed over the protein surface, but binds with several well-defined water binding sites. In addition almost all water molecules are bound to one side of the protein, while the other side of the protein is almost completely dry. Unexpectedly the binding site of the protein and its active site are almost completely dry.

The effect of increasing the water activity (by adding more water molecules into the system) was also studied. One Molecular Surface Map was created for each simulated system, which offered the possibility of easily comparing the simulated systems. As expected, with increasing water activity in the simulation system, the amount of water

molecules on the protein surface increases. The majority of the added water molecules still bind to the already identified high affinity water binding sites. However, additional patches on the protein surface were identified to which water binds only during very high water activity. These new results can help to better understand the molecular mechanism of enzymes, which can be used to develop strategies to improve the efficiency of proteins in the laboratory.

The cases above are just examples of the usefulness of the Molecular Surface Maps in analysing the function of the molecular interface. More examples, such as time-dependent data, comparisons of similar proteins and an investigation of protein architecture are outlined in the original paper [86]. Overall, the results demonstrate that the maps are a useful addition in establishing molecular visualisations and are helpful when striving to gain further insights. They provide easy access to the shape as well as the physico-chemical properties of the molecular surface by showing both in a two-dimensional representation. This representation solves the typical problems of the three-dimensional surface such as view-dependency and occlusion.

## 3.2 Clustering

Numerous methods have been developed to compare proteins with each other. La et al. and Xiong et al. [91, 172] describe their 3D-Surfer software that uses 3D Zernike descriptors as described by Seal et al. [130] to extract feature vectors for each protein to achieve comparisons of large protein numbers. They do not incorporate further biochemical properties of the surface that may be important to determine the function of a protein. Bock et al. [18] describe an approach to compare areas on protein surfaces utilizing so-called spin images. Their computationally complex approach also only utilizes geometrical information but they suggest incorporating physico-chemical properties. As proposed by Anzali et al. [8], the shape of one molecule can be stored in a neural network to achieve direct comparison with others. Such direct comparisons are also possible through the use of gradient vector flow [138], or, again by 3D Zernike descriptors [129]. All of the aforementioned methods have in common that fact of being computationally relatively complex. Hofbauer et al. [62] construct representatives in the form of graphs and then compare the resulting graphs with each other. They also compare physico-chemical properties, but their method, again, suffers from computational complexity, as the comparison between two proteins alone can take up to several minutes.

Besides this purely biological application, the hierarchical clustering of images has been researched extensively. A review of commonly used clustering algorithms has been given by Saxena et al. [137]. The K-Means algorithm, probably first described by Steinhaus [152], is one of the most well-known and widely used clustering algorithms.



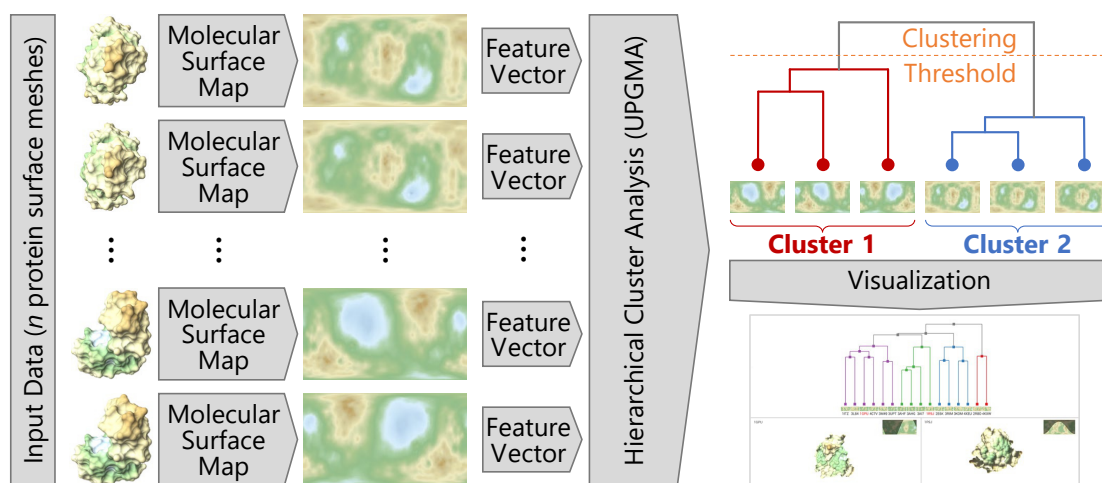
As it requires the knowledge of the number of clusters beforehand, it struggles to solve many real-world application cases where this number is unknown. Nonetheless, it is used by the methods of Cai et al. [25] and Pandey and Khanna [114]. The first method uses image features alongside textual features to cluster images found in the world wide web. The second method agglomeratively clusters images by consecutively calculating representative images of the clusters and using them for further clustering.

One of the most well-known clustering applications in biology is the phylogenetic analysis. There, biologists attempt to understand the evolutionary relationships between different proteins or genes. The result of this type of clustering are so-called phylogenetic trees [40], often visualised by a dendrogram. Several methods for their construction exist, currently the most widely used is the neighbour-joining method, first described by Saitou and Nei [131]. As opposed to the Unweighted Pair Group with Arithmetic Mean method (UPGMA [150]) they do not necessarily produce rooted trees and do not assume a constant rate of evolution. Like all other methods constructing phylogenetic trees, the resulting visual layouts of these methods are often ambiguous, depending on the order of the input. To examine the differences in phylogenetic trees, Bremm et al [21] present a comparison approach. As the resulting tree structures become quite large, Huson et al. [68] describe a visualisation system for their efficient display.

We introduce a hierarchical clustering of the Molecular Surface Maps, discussed in the previous section. The clustering is based on the idea that visually similar maps also imply a similarity in the function of the underlying protein. The hierarchical clustering is visualized as a dendrogram, as shown in Figure 3.11. We compute a unique descriptor for each map that allows to identify similar maps even if the content is translated, rotated, or scaled. The distance between a pair of descriptors is used as a measure of similarity.

### 3.2.1 Method

Our algorithm overcomes some of the aforementioned deficits and consists of four steps as illustrated in Figure 3.11. In the first step the proteins are aligned using either the Root Mean Square Deviation (RMSD) method or an approach that is based on Principle Component Analysis of atom positions. In the second step three Molecular Surface Maps, as described in subsection 3.1.1, are created for all proteins in the input data set, each contains the scalar values of the associated surface property, see the greyscale images in Figure 3.12. The first of the three properties is the colour coded heightmap of the topological structure as described in subsection 3.1.2. The second is the hydrophobicity, which describes how energetically (un-)favourable an interaction with water molecules would be and the third is the temperature factor, which is an indicator for the flexibility of the protein. In the third step, a feature vector for each generated map is computed,

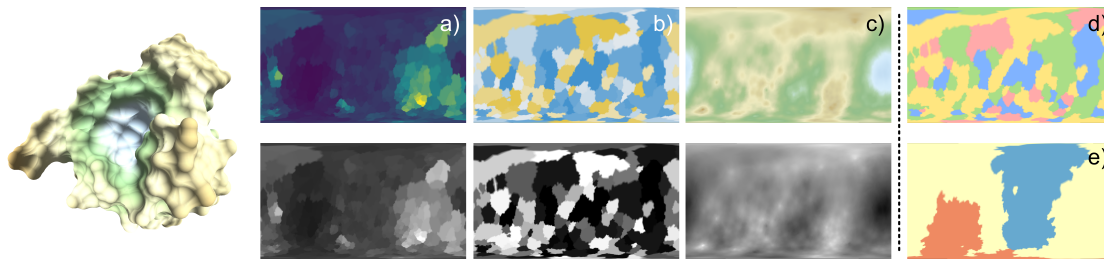


**Figure 3.11** — Schematic overview of the hierarchical protein clustering approach (from left to right). The input is an ensemble of proteins. For each protein, a three-dimensional molecular surface representation is computed, which is subsequently transformed to a two-dimensional Molecular Surface Map [86]. From each of these maps, a descriptive feature vector is extracted using either Image Moments, Color Moments, or a Convolutional Neural Network. Based on the distances between these feature vectors, a hierarchical clustering is computed using the UPGMA algorithm [150]. This clustering is then displayed in an interactive visualisation application, alongside surface visualisation of user selected proteins for comparison. © EG 2020 [141].

resulting in three feature vectors for each protein. They can be concatenated to a vector containing two or three of the original vectors. The features can be computed by three different methods: Image Moments [64, 41], Color Moments [102], and a Convolutional Neural Network (CNN) [133]. For the Color Moments a colour mapping from the scalar values to RGB values (see Figure 3.12) was used. The other two methods work directly on the scalar maps. In the fourth step, the distances between the feature vectors are used to cluster the proteins hierarchically. Optionally, the feature vector computation and clustering can be executed on individual tiles of the molecular surface map instead of on the whole image. The resulting clustering can be interactively explored using the MegaMol visualisation framework. It shows the clustering alongside the SES visualisation of user selected proteins for a direct comparison (cf. Figure 3.11 bottom right).

**Alignment** In the first step of the algorithm all proteins are aligned. The visual appearance of the Molecular Surface Maps depends greatly on the orientation of the





**Figure 3.12** — Overview of the different maps for the PYR domain of a transketolase from *Pseudomonas aeruginosa* (PDB ID: 4XEU), which is shown to the left as molecular surface. The colour-coded maps (a – c) in the top row were generated using the scalar maps that are shown in the bottom row (black and white maps). The two maps to the right are based on nominal data and thus have no value maps. The maps display: (a) B-factor colouring, using the *Viridis* colour map (purple to yellow). (b) Hydrophobicity interpolated between blue (hydrophilic), white (neutral), and yellow (hydrophobic) (blue to yellow). (c) Cartography-inspired colouring by elevation, i.e., the distance in Å between the surface and the centroid of the protein (blue to yellow). (d) different physico-chemical properties of amino acids: basic (blue), aromatic (red), polar (green), or unpolar (yellow). (e) colouring by contact areas where the surface of the PYR domain (yellow) is in contact with the two other domains (PP domain orange and TKC domain blue). The values of the scalar maps have been normalized for improved visibility. © Elsevier 2021 [140].

protein. Even for very similar proteins, however, the orientation can differ widely in the Protein Data Bank [14]. Therefore, it is advisable to align the proteins as accurately as possible prior to the mapping. For similar proteins, minimizing the Root Mean Square Deviation (RMSD) between corresponding atoms is the most commonly used alignment method [75]. However, if the input data is highly heterogeneous, this approach is not feasible, as it is not possible to establish the necessary per-atom correspondence for highly different proteins. For these cases a more general approach based on Principle Component Analysis of the atom positions can be used. Each protein is rotated so that the first principle component is aligned to the x-axis and the second principle component is aligned to the y-axis. Thus, proteins with a similar shape will be oriented similarly in a fast and convenient way, regardless of the underlying chemical sequence. This works well for elongated as well as globular proteins, as possible flipping in either x- or y-direction is later handled by the rotationally invariant feature extraction methods. Nonetheless, as the established RMSD method is more accurate, it should be used on all data sets wherever possible (e.g., similar domains).

**Map Creation** The next step after the alignment is to create the three maps for each protein using the Molecular Surface Map algorithm as described in subsection 3.1.1. Since the clustering needs the scalar values, the algorithm was changed so that the map contains the scalar values of the property directly. That is, suitable colour scales can be applied afterwards. Due to the high number of proteins the tunnel detection uses the fast iterative method instead of the combination of AO and the Reeb graph. In case the fast detection fails, the slower but more accurate, method is used. This, however, occurred in less than 10% of the tested proteins. The combination of different methods, as well as different probe sizes, due to the iterative method, may lead to the comparison of surfaces generated for different probe radii. As a change to the probe radius only affects small local concave parts of the surface, the effect on the resulting surface is minimal. While the map for most colouring modes can be derived directly from properties of the depicted atoms or amino acids, the contact map (cf. Figure 3.12e) requires an additional pre-computation step. Contact maps are computed by calculating atom-atom distances between the viewed protein and one or more contacting molecules. Where the minimum distance lies under a certain user-defined threshold (with a default value of 5 Å in most cases), the surface colour is changed to the colour assigned to the contacting molecule. If two or more atoms are in contact, the map is coloured according to the closest one.

**Feature Computation** To cluster the Molecular Surface Map images generated in the previous step, a descriptive value is assigned to each. Three approaches were used: Image Moments, Color Moments, and a feature computation based on a Convolutional Neural Network (CNN). Each method calculates a feature vector—with seven, nine, and 1792 elements respectively—to uniquely represent the image. For both, the Image Moments and the CNN features, the calculated maps are used as input. To be able to properly use Color Moments, a colour map has to be applied to the scalar maps first. The quality of the result heavily depends on the quality of the used colour map, but otherwise, the feature vector would not be long enough to accurately represent the image. Using either of these approaches, each Molecular Surface Map computed in the first step is assigned a feature vector. To create a more unique and meaningful feature vector, multiple Molecular Surface Maps representing different quantities of the same protein can be combined by simply concatenating the feature vectors. Concatenating the feature vectors can, for example, take mutations of the proteins into account that only alter certain quantities.

Image Moments are derived from the intensity value of a pixel  $I(x, y)$ . Therefore, the previously calculated Molecular Surface Maps can be used directly. Computing the Hu moments invariants ( $I_1, I_2, I_3, I_4, I_5, I_6, I_7$ ) [64] leads to invariance under translation, rotation and scaling. As proposed by Flusser [41] the moment  $I_3$  was replaced by  $I_8$  as it is dependent on the other moments. This results in a feature vector  $F_{im}$  for each

Molecular Surface Map containing seven moments:

$$F_{im} = (I_1, I_2, I_4, I_5, I_6, I_7, I_8)$$

Color Moments are computed as proposed by Maheshwari et al. [102] using the colour-coded, and not the scalar value, map as input (see Figure 3.12 top row). The first three elements in the resulting feature vector are the mean values of each RGB colour channel ( $E_{RGB}$ ). The next three components of the feature vector are the standard deviation, again for each colour channel ( $SD_{RGB}$ ). The final three components represent the skewness for each colour channel separately ( $S_{RGB}$ ). This can be understood as a measure of the degree of asymmetry in the distribution. The combined feature vector  $F_{cm}$  for the Color Moments contains nine elements and uniquely describes the associated map.

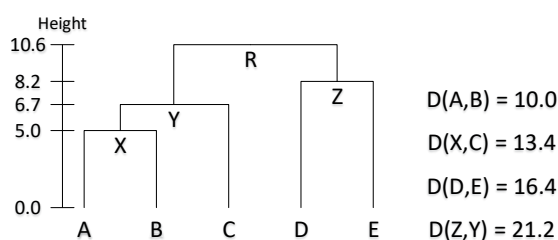
$$F_{cm} = (E_R, E_G, E_B, SD_R, SD_G, SD_B, S_R, S_G, S_B)$$

The third feature computation method is a CNN called MobileNetV2, which was developed by Sandler et al. [133] and is provided as a TensorFlow [1] module. Due to its modular structure, the authors provide a smaller version of the network that only calculates image feature vectors, which can be used as input features for the subsequent calculations. Compared to the other two feature extraction methods, the feature vectors retrieved from MobileNetV2 are significantly longer. For each map a feature vector  $F_{nn}$  with 1792 elements is obtained, which is invariant under translation, rotation and scaling:

$$F_{nn} = (F_0, F_1, \dots, F_{1791})$$

**Clustering** In the final step of the algorithm a hierarchical clustering of the  $n$  input proteins based on the feature vectors is computed. In order to get the  $n \times n$  distance matrix, the Euclidean distance  $d(x, y)$  between each pair of feature vectors is calculated. The resulting distance matrix is then used in the iterative process to compute a binary tree containing all maps or proteins respectively. For each of the proteins, a leaf node is created containing the feature vector and the corresponding Molecular Surface Map. Initially, each leaf node is regarded as a one-element cluster. The further construction of the tree follows the UPGMA algorithm developed by Sokal and Michener [150]. In bioinformatics, UPGMA is widely used to construct phylogenetic trees. Their algorithm follows a bottom-up approach by subsequently merging the clusters with the shortest pair-wise distance. After each merge, the distance matrix is updated by removing the two merged clusters and adding the newly created cluster instead. New distance values from the new cluster  $C_N$  to all existing ones  $C_i$  are calculated as follows:

$$d(C_N, C_i) = \frac{1}{|C_N| \cdot |C_i|} \sum_{x \in C_N} \sum_{y \in C_i} d(x, y)$$



**Figure 3.13** — UPGMA algorithm [150] for five nodes, A to E. The complete distance matrices of each iteration are not shown, but the shortest distances of each iteration are shown on the right side of the tree. The height of the connection is computed as  $h_0 = D_0 / 2$  for the first level and  $h_i = D_i / 2 - D_{i-1}$  for the other levels. © EG 2020 [141].

Therefore, the new distance value  $d(C_N, C_i)$  is equal to the arithmetic mean of all pairwise node distances. The new cluster is assigned a descriptive feature vector as well, which is the centroid computed from the feature vectors of all leaf nodes contained in the two merged sub-trees. The representative map of the cluster is the map of the leaf node in the sub-tree of  $C_N$  with the most similar feature vector to this centroid. After the iteration finishes the binary tree contains all maps that provide a preliminary hierarchical clustering. The UPGMA algorithm is able to generate trees where the distance in the tree following the edges corresponds to the actual distance in the distance matrix (see Figure 3.13). This is achieved by assigning a height value that is equal to half the distance of the sub-trees to each newly created cluster node. Constructing the tree in this way allows for an intuitive visual analysis, as the distance values can be estimated directly from the visualisation. Using a user-defined similarity threshold  $T_c$  the final clusters based on the UPGMA tree can be determined. All nodes that belong to a sub-tree of less than height  $T_c$  belong to the same cluster (dashed orange line in Figure 3.11).

It is important to mention that the clustering is only possible for map types where the values lie at least on a ordinal scale. If a map contains only nominal data, a distance function to perform the clustering cannot be formulated. As the physico-chemical maps (d) and the contact maps (e) in Figure 3.12 display nominal data, they are only used for display and analysis purposes, not for the generation of the clustering. The physico-chemical maps display nominal data, as they show the most prominent physico-chemical property for each amino acid. That is, these properties are categorical (four different categories) and the categories do not lie on a common scale, which leads to a nominal map.

**Tiled Clustering** Especially for proteins that contain local distortions, but overall exhibit the same surface structure and properties, comparing the maps as a whole can lead to a larger global distance even though the function is very similar. For such cases a tile-based approach was developed that subdivides the maps into overlapping

smaller images, which allows for a local-to-global comparison. That is, the small tiles are compared individually, resulting in a global distance between two maps which is not necessarily the same as the one calculated by a comparison of the whole maps.

The approach works as follows: the value images are subdivided into smaller overlapping images of fixed size. To obtain the highest accuracy, the overlap should be as large as possible as the viewed feature might also be contained in the other map, just moved by few pixels. The implementation uses an overlap of 50%, as it is a compromise between computational cost and accuracy. The size of individual tiles can further depend on the use case and, therefore, on the size of the displayed proteins. Generally speaking, it is preferable to keep the tiles small enough to avoid the aforementioned problems, and big enough to contain at least the footprint of one amino acid. As a default value, a tile size of  $160 \times 160$  px is used, which fits to the default map resolution of  $2560 \times 1440$  px and the average enzyme size. After all tiles have been extracted, feature vectors are determined for each of them individually. Then, for each tile of one map, the feature vector distance to each tile of the other map is calculated. Out of these distances, the smallest one is chosen as representative distance for the tile. The final distance between the two maps is then calculated by averaging the thus determined distances. While this approach can lead to more accurate results, it obviously has a significantly higher computational cost than comparing whole maps. Due to the increased computational complexity, the suggestion is to use it only on small data sets, i.e. with less than 100 proteins.

### 3.2.2 Results and Discussion

The approach was tested using two types of protein ensembles. The first type consists of proteins retrieved from the Protein Data Bank [14], which were selected so that they have previously known properties. It is used to evaluate the correctness and feasibility of the proposed method. The second type is an ensemble of functionally similar proteins that was provided by domain scientists.

The first data set, which consists of ten proteins, was constructed so that it would contain two clusters and one outlier. This was achieved by selecting three sufficiently different proteins—or, more specifically, enzymes. For two of these three proteins, a sequence-based similarity search was performed. Out of these search results we picked the most similar proteins to be included into the cluster. The results are shown in Figure 3.14. All three extraction methods were tested using different Molecular Surface Maps and combinations, as shown in Table 3.2. The ground truth for this evaluation is based on information about the functional similarity of enzymes provided by the Enzyme Classification (EC) [167].

Using MobileNetV2, the two clusters and the outlier are generally well preserved (Figure 3.14(a) and (b)). The one exception is the b-factor map in Figure 3.14(c), where

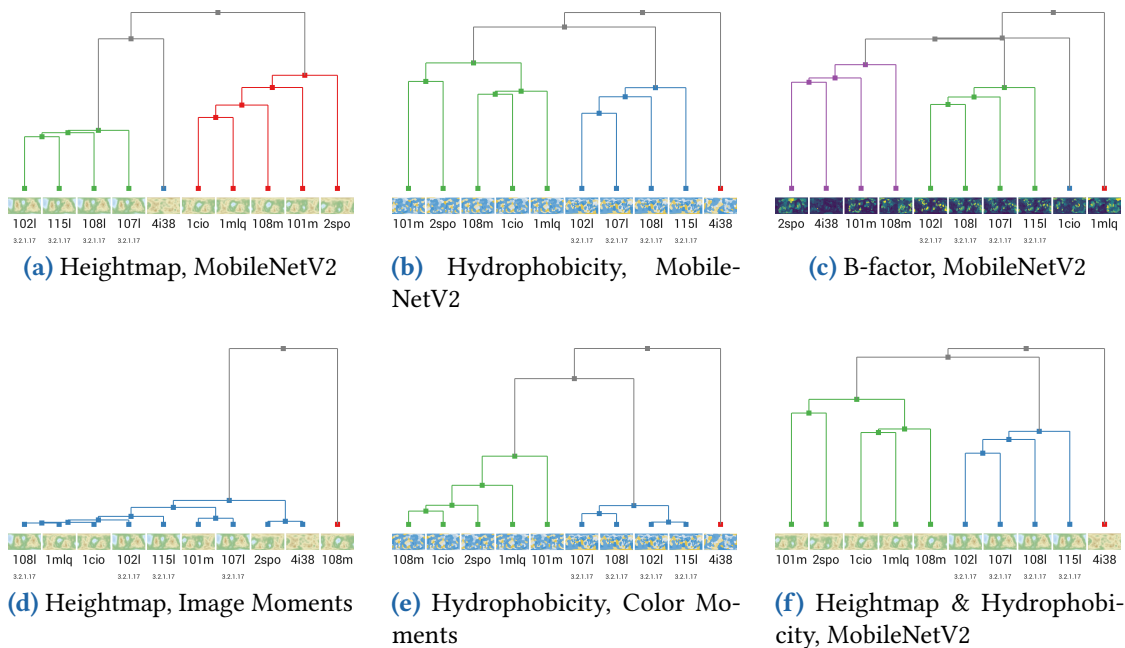
**Table 3.2** — Colour-coded results of the clustering for the evaluation data set using each of the three feature extraction methods: Image Moments (IM), Color Moments (CM) and MobileNetV2 (MN) and different maps: Heightmap (H), Hydrophobicity (Y), B-factor (B) and the combinations H-Y and H-Y-B. The expected result is two clusters, C1 and C2, as well as one outlier O. Configurations marked blue resulted in a proper clustering, red configurations were not successful. Yellow values mark borderline cases, where an incorrect protein was added to a cluster at a later point in time, indicating a large distance to the rest of the cluster. For H-Y and H-Y-B the feature vectors of the maps are combined, either without or with the B-factor features, respectively.

	IM			CM			MN		
Map	C1	C2	O	C1	C2	O	C1	C2	O
H	-	-	-	+	+	o	+	+	+
Y	+	+	+	+	+	+	+	+	+
B	-	-	-	-	-	+	+	-	-
H-Y	+	+	+	+	+	o	+	+	+
H-Y-B	-	-	-	-	-	+	+	+	+

only one cluster is preserved properly. This is also true for the other feature extraction methods, Image and Color Moments. Comparing the resulting clustering trees of the different methods shown in Figure 3.14, it is noticeable that Image Moments and Color Moments tend to have a more diverse distance value distribution than the CNN features. This means that the distance between the closest proteins is far smaller compared to the distance between the farthest ones. Therefore, many merges happen at the lowest point of the clustering tree as opposed to the MobileNetV2 results. This can be attributed to the length of the feature vectors. As the feature vector given by MobileNetV2 has 1792 entries, even small distances for single vector elements add up to larger numbers. Although the base difference values are typically high, the quality of the results is superior to the other two methods.

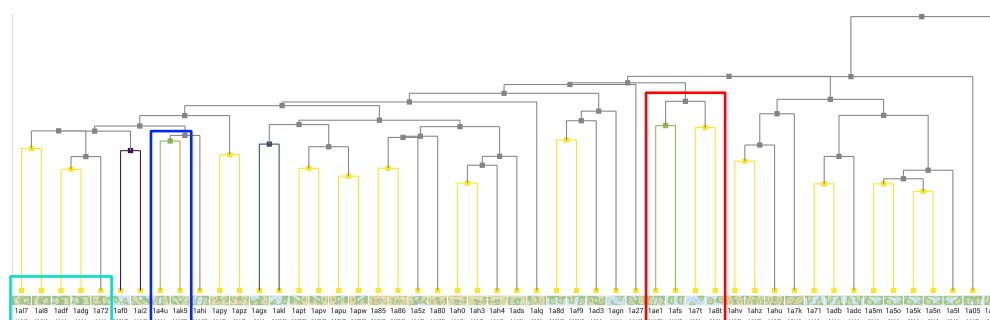
As described in section 3.2 and shown in Figure 3.12 three different map types were chosen to represent different properties of the proteins: the heightmap represents the geometry of the surface, the hydrophobicity can affect binding behavior, and the b-factor is an indicator for flexibility. Initially the b-factor was considered to be an interesting feature for the function of a protein, as it can influence the catalytic rate. However, as it is not decisive for the function and it can change under different environmental conditions, it is omitted for the larger data sets due to the poor results for the small ensemble. Only in very specific cases, such as when investigating changes in the catalytic rates of an ensemble of similar enzymes (e.g., different mutants of an enzyme), a clustering based on flexibility could be interesting, as it can influence the catalytic rate. For the other



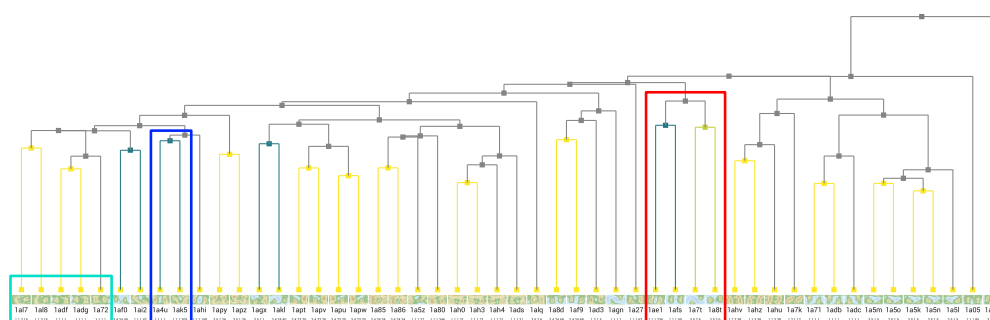


**Figure 3.14** — Comparison of several similarity trees of the ten protein data set. The subcaptions list the colouring mode alongside the used feature extraction method. All subfigures display, from top to bottom, the generated tree, the maps belonging to each node, their respective PDB identifier, and, if available, their enzyme classification. Proteins with a displayed enzyme classification should form one cluster, the remaining proteins another one. 4l38 is considered an outlier. Using MobileNetV2 features the clustering method generates the expected result fully ((a) and (b)) or at least partially (c). Color Moments (e) leads to similarly good results. In contrast, Image Moments shown in (d) completely fails to categorize the proteins similarity properly. Finally, (f) shows the combination of two map types that mutually alleviate the shortcomings of each single map type. © EG 2020 [141].

two map types, the clustering quality depends on the used feature extraction method. Image Moments were successfully used by Kolesár et al. [81] to calculate similarities for maps of protein cavities. In this use case, however, they performed worse than Color Moments and MobileNetV2. They even considered one of the enzymes as a clear outlier while merging the actual outlier early. The assumption is that Kolesár et al. obtained good results since their maps consisted of large distinct patches that differed only slightly between the maps. Although Color Moments strongly depend on the chosen colour map, they performed considerably better than Image Moments. However, they exhibit weaknesses in the detection of outliers (cf. Table 3.2). MobileNetV2 produced the overall best results, especially when considering that the results of using Color



(a) Direct merges coloured by enzyme classification



(b) Direct merges coloured by TM-score

**Figure 3.15** – Clustering of 50 different proteins using MobileNetV2 feature vectors for heightmap and hydrophobicity. Visually similar maps are clustered together. Merges of two leaf nodes are coloured by the quality of the merge based on two different external scores or classifications. Direct matches are coloured using the *Viridis* colour map (■), where yellow corresponds to a good match and purple to a poor one. (a) uses a given enzyme classification as ground truth, (b) the TM-score [178]. The cyan box (■) contains five enzymes belonging to the same class, where the TM-score as well as the clustering approach detect a high similarity. The red (■) and blue (■) boxes are cases where the clustering approach gives a more suitable similarity score than the TM-score. © EG 2020 [141].

Moments features might worsen when choosing a different colour map.

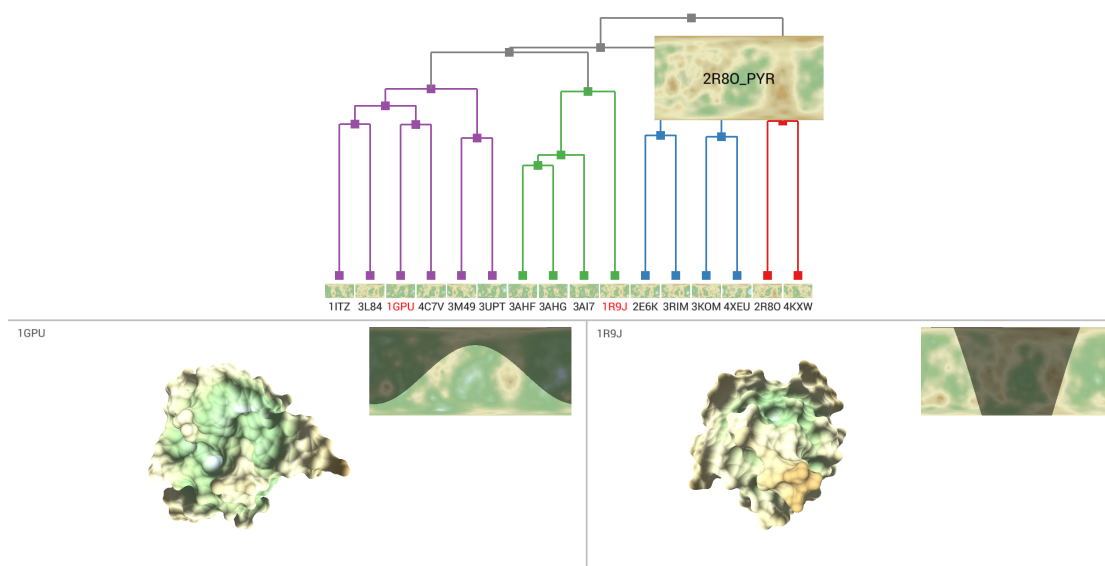
As mentioned in section 3.2, the algorithm can also use a combination of multiple maps for each protein for the clustering. This can be useful since surface properties like the hydrophobicity as well as the shape of the surface influence the function of a protein. As shown in Table 3.2 and Figure 3.14(f) the MobileNetV2-based clustering produces very good results. Since MobileNetV2 performed best on our small verification data set, only this method and the combination of heightmap and hydrophobicity map is used for the larger data set.



Figure 3.15(a) shows results for the ensemble of 50 proteins. This data set consists of short, randomly chosen sections of enzymes, in which each section consists of enzymes with a similar EC classification. That way, it can be ensured that the resulting set of 50 proteins contains multiple clusters of enzymes with similar function. When inspecting the results of the clustering, it can be observed that visually, as well as functionally, similar enzymes are linked together early, e.g., as seen in the cyan box. These proteins, which were identified as being similar by the clustering, are also classified as being functionally similar according to the EC classification. Overall, the majority of the clustering results for enzymes are consistent with the EC classification. That is, proteins of the same class, and therefore, same function are in the same cluster. Two outlier pairs are visible, 1AGX and 1AKL, and 1AF0 and 1AI2. The first pair actually belongs to the same main class and their shape is at least roughly similar, which explains the clustering. That is, the remaining pair (1AF0 and 1AI2) is the only real outlier, as their heightmaps are different and they are not in the same EC class.

Figure 3.15(b) shows a comparison of results with the TM-score (Template Modeling score) [178] for the aforementioned ensemble of 50 proteins. The TM-score is a widely used protein similarity measurement score (see, e.g., [47, 51]). It compares the amino acid sequence and 3D structural information of two proteins and returns a single similarity value, which can be seen as a global fold similarity score. While a generally very good consensus between our method and the TM-score (yellow nodes) can be observed, the clusters in the blue and red boxes are especially interesting when we compare these against the TM-score values. Here, the TM-score indicates a high dissimilarity between these proteins although they are functionally very similar according to their EC classification. With our method, these cases are clustered early, but typically later than cases where all four enzyme classes match. Additionally, the TM-score was not able to catch the similarity of 1A7T and 1A8T completely, although they fall into the same class (right half of red box). Although the heightmaps do not look very similar, our method was able to correctly cluster them. Failed similarity detections by the TM-score can happen when the interior of two proteins is different but the surfaces are similar. Our method will then detect a high similarity while the TM-score indicates a high difference.

The cases above are just examples of the usefulness of the clustering in finding proteins with not only a similar structure but also a similar function. Further examples, such as an analysis of protein domains, are given in the original paper [140]. Overall, the results demonstrate that the approach can be used to cluster and visually analyse previously unknown proteins and hypothesize about the function of the protein. The hierarchical clustering provides a good overview and the 3D views help to assess the quality of the map-based similarity score.



**Figure 3.16** — Overview of the visualisation application for analysing the result of the clustering. The top view shows the hierarchical binary tree, coloured according to the clustering. In the two detail views at the bottom, the molecular surfaces of the protein corresponding to the selected maps are visualised for further analysis. The names of the selected proteins are marked in red. Each surface view contains an inset that shows the corresponding map which is highlighted in the area the user is currently looking at on the 3D surface. Both the maps and the surface are displayed using the heightmap colouring. The side of the surface that is averted from the viewer is darkened on the map, while the side facing the viewer is highlighted to provide a spatial reference. Note that the two detail views are unlinked in this example, so that the camera can be adjusted individually for each view. © Elsevier 2021 [140].

### 3.3 Summary and Conclusion

The planar representation for molecular surfaces and their respective attributes discussed in this chapter, not only allows researchers to analyse a single protein but also compare, analyse and cluster protein ensembles based on the similarity of their surface. The Molecular Surface Map provides a view-independent, occlusion-free representation of the molecular surface, which shows the topography in addition to the physico-chemical properties. In addition to the physico-chemical properties other properties can be shown on the map using different colouring schemes. One example of this would be the aggregated solvent binding that shows where on the surface, water molecules bind. Another example would be the highlighting of binding sites, which

allows the computation of a Space-time Cube in order to determine the accessibility of one or more binding sites during a molecular dynamics simulation. Additionally, the maps can be used to hierarchically cluster and analyse protein ensembles based on surface similarity. The surface similarity takes the topography as well as different physico-chemical properties into account, i.e. multiple maps per protein. Therefore the Molecular Surface Maps can be used to interactively explore and analyse either a single protein or an ensemble in order to gain new insights. The prototypical visualisation application (cf. Figure 3.16) can be used to interactively explore and analyse the hierarchical clustering. This allows expert users to investigate the results in the dendrogram and interactively set the threshold determining the clusters. A detailed analysis of individual pairs of proteins is possible in the 3D views, which helps to assess the quality of our map-based similarity score. Consequently, the application can be used to cluster and visually analyse previously unknown proteins and hypothesize about the function of the protein.

Both methods benefit from being rendered on a large high-resolution display. For example, the larger screen space allows to show multiple different Molecular Surface Maps at once in order to analyse and compare them interactively. Another example would be to show multiple Molecular Surface Maps of the same protein, but using different colour modes to show different properties, or to show Molecular Surface Maps from different timestamps. The same is true for the clustering of larger datasets, as the resulting UPGMA tree becomes very large. Again the larger screen space helps with the visual analysis of the results, since exploring the dendrogram for larger ensembles of proteins is a tedious task on regular displays. Additionally, this provides the opportunity for collaboration as multiple users can interact in front of the large display and analyse the results. While the two methods can benefit from being rendered on large display, this is also true for molecular dynamic simulations and other large scientific datasets.

Both methods discussed in this chapter were developed in close collaboration with domain experts to make sure that they are useful to them and that they satisfy their requirements. The successful application of these methods and the resulting discoveries discussed in this chapter reflect the feasibility of this collaborative approach.



## REAL-TIME HIGH-RESOLUTION VISUALISATION SYSTEM

### Parts of this chapter have been published in:

- F. Frieß, C. Müller, T. Ertl. Real-Time High-Resolution Visualisation. Proceedings of the Eurographics Symposium on Vision, Modeling, and Visualization, pages 127–135, 2020.. [46]
- F. Frieß, M. Braun, V. Bruder, S. Frey, G. Reina, T. Ertl. Foveated Encoding for Large High-Resolution Displays. IEEE Transactions on Visualization and Computer Graphics, pages 1–10, 2020. [44]
- F. Frieß, M. Becher, G. Reina, and T. Ertl. Amortised Encoding for Large High-Resolution Displays. IEEE 11th Symposium on Large Data Analysis and Visualization, pages 53–62, 2021. [43]

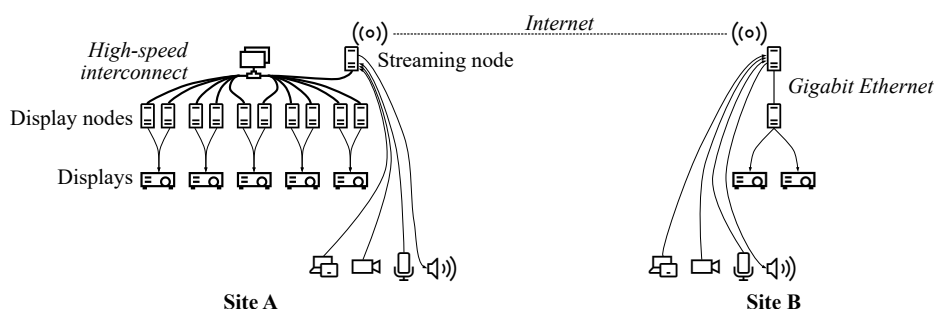
Large displays are already suitable for collaboration and additionally provide benefits for the interactive exploration of scientific datasets, as shown in section 3.3. Therefore, users would clearly benefit from being able to collaborate remotely using such display scombining video and audio conference setups with the possibility of sharing high-resolution interactive (scientific) visualisations. However, while large – often tiled – displays with image resolutions beyond 4K require a system that can handle a diverse combination of software and hardware setups. While previous work, e.g. SAGE 2 [103, 123], moved away from pixel streaming and focused more on supporting diverse hardware setups, the system described in this chapter shows that pixel streaming is

now a viable approach for interactively sharing the content of a large high-resolution display.

The system makes use of modern programmable GPUs to accelerate not only the computations but also the video encoding and decoding. Its focus lies on real-time processing of video and audio streams, enabling the content of a large high-resolution displays at high frame rates to be shared. By exploiting recent advances in networking technology and the wide availability of fast, low-latency hardware video encoders, the system is based on pixel streaming in combination with hardware video encoding and decoding as the solution for sharing interactive visualisations in such environments – not least because it has even found its way into gaming, which notoriously craves speed and low latency [146, 27], in the form of commercially available products like NVIDIA’s GeForce Now, Sony’s PlayStation Now, Google’s Stadia and Microsoft’s upcoming project xCloud. Even streaming stereo images to a stereo-capable large tiled display has been explored [100].

The system described in this chapter grew out of a large collaborative research project<sup>1</sup> in visual computing involving two universities – the University of Stuttgart (site A) and the University of Konstanz (site B) – about a two-hour drive apart. Typically, there would have been a lot of travelling involved in attending seminar talks, PhD colloquia, project demonstrations, etc., alternating between the sites. Since each site already had a large Powerwall-style visualisation system (with 44 Mpx and 11 Mpx respectively) installed in a lecture room environment, it was clear that these large displays could be used for as many presentations as possible. Figure 4.1 illustrates the original usage scenario and the different hardware setups involved. While one of the Powerwalls uses multiple GPUs in a single machine to drive the whole wall, the other one has a distributed framebuffer, such that even showing Powerpoint slides from a laptop is a non-trivial task. Software for this system is therefore custom-made and not necessarily portable to other installations. Thus, the desire to share the content of a Powerwall in such a way that interactive high-resolution visualisations as well as presentation slides and 4K camera streams would be visible on the other side and combined with high-quality audio stream to allow for collaborative seminars and demonstrations without the need for many people to travel (cf. Figure 4.2). The system has been in productive use for three years now. It uses the aforementioned state-of-the art hardware video encoders to transmit pixel streams in real time. It also offers a low-latency solution to share the frames generated by any application on a large high-resolution display with one or multiple other large high-resolution displays, regardless of the different hardware setups and without changing the code of the application. The same technology is also used to capture and transmit content from devices such as laptops, which enables users to share their presentation slides without having to install any additional software,

<sup>1</sup> SFB-TRR 161 “Quantitative Methods for Visual Computing”, <https://www.sfbtrr161.de/> (last accessed 22/04/2022)



**Figure 4.1** — Conceptual overview of the two connected Powerwalls for which the system was originally developed. The high-resolution display on the left-hand side uses a cluster of computers to drive its projectors. Like in a HPC cluster, the nodes are connected using a high-speed, low-latency InfiniBand network. In contrast, the display on the right-hand side uses a single computer with multiple outputs to drive the displays. In both cases, a dedicated streaming node is tasked with collecting and distributing all audio and video signals, although this role could be delegated to another node as well. © EG 2020 [46].

enabling a standard-projector experience when giving a talk, and video cameras using a resolution of 4K. In order to account for different usage scenarios, audio streams, video streams and desktop sharing streams can be configured and arranged as needed.

Research for ways of supporting collaboration across sites using multiple wall-sized displays lead to a multitude of systems, frameworks and libraries. The two, probably most widely known, collaboration environments for display clusters are the Scalable Adaptive Graphics Environment (SAGE) [124] and its successor, the Scalable Amplified Group Environment (SAGE2) [123, 103]. While SAGE is also based on pixel streams it requires code changes to the applications in order to generate these streams. The newer SAGE2 is a browser-centric reimplementation around a central Node.js-based web server, which is responsible for distributing the shared content to different clients. This way, it naturally supports any client ranging from desktop computers through laptops to smartphones as long as a recent browser is available. In order to support high-resolution displays driven by graphics clusters, SAGE2 provides the means to synchronise the page redraw on connected browsers. Although this approach works for any device it does not offer the performance required to share the content of a large high-resolution display in real time. Klapperstueck et al. [77] developed a system for interactive collaboration with mobile devices and displays of various sizes. It allows users on different sites to share content to a virtual desktop that can be simultaneously displayed on multiple large displays. Users can also arrange and annotate the content they share using the developed client software that can be adapted with plug-ins for existing data analytic software. CubIT [127] is designed to handle multiple multi-touch screens and creates





**Figure 4.2** — The conference scenario with a talk given at site A. The tiled display shows the speaker’s slides captured from the laptop on the left and the 4K video stream from site B on the right. Site B receives the slides and a 4K video stream from site A, while communication between the sites is established by bidirectional audio streams. © EG 2020 [46].

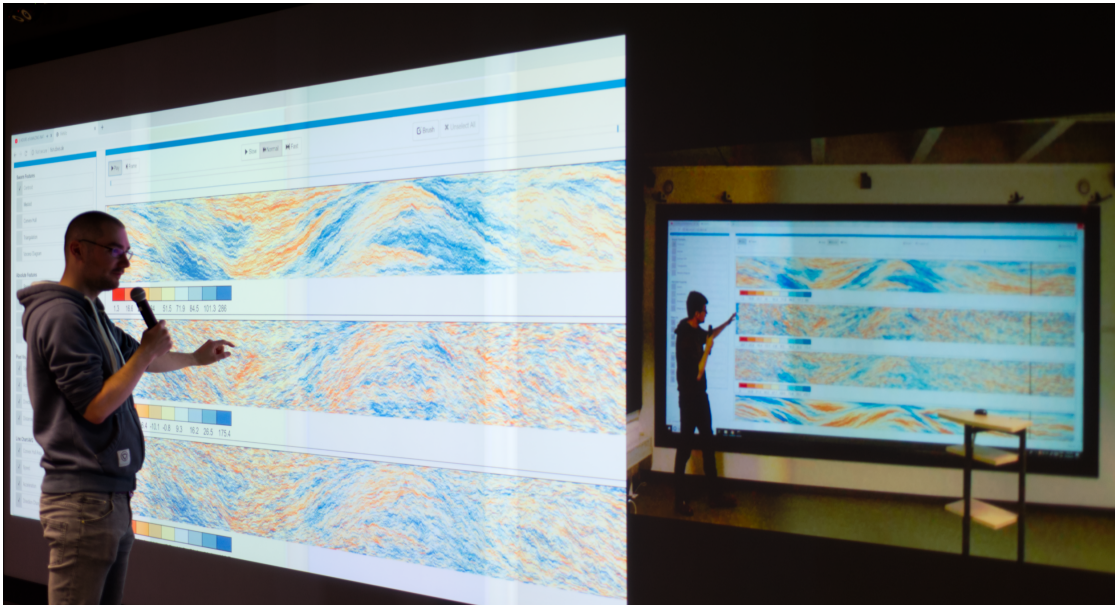
a shared workspace across these screens. Like SAGE2 they use a web-based content platform to upload and moderate content, such as images, videos and text. Interaction with the content is implemented via drag and drop and a mobile interface that allows users to create content and send it to CubIT for display on the tiled display. This allows for on-site collaboration and content management. The interactive visualisation environment DisplayCluster by Johnson et al. [74], and its successor Tide [17], target large tiled displays driven by clusters. Their system is able to handle multiple input devices and can show content from remote sources by streaming the content from these devices. Additionally, it also supports the viewing of media files directly while distributing the workload of processing and rendering the content to the cluster, thus achieving better scalability with the display size. Like SAGE, DisplayCluster contains a window management system allowing for content to be arranged. LACOME [98, 101] allows users to simultaneously show multiple computer desktops on a shared large screen display by using one or more Virtual Network Compute (VNC) servers. The interaction with the large screen display is realised by forwarding the individual user’s mouse and keyboard input to the shared display space. This allows users to interact with the windows and content on the display. Beck et al. [12] presented a multi-user immersive telepresence virtual reality system that allows users to meet in a shared virtual 3D world. It provides users with perspectively correct stereoscopic images and 3D information about the local workspace.

In order to display content on large high-resolution displays, a variety of parallel



rendering frameworks and middleware have been suggested. The Cross Platform Cluster Graphics Library (CGLX) [34, 116] is an OpenGL-based framework for distributed high-performance visualisations. It allows users to adapt existing, or develop new, OpenGL-based applications for clusters such as tiled displays. Additionally, CGLX supports co-located collaboration through multiple multi-touch devices to which the updated scene information is streamed. Compositing the displayed frame in a distributed rendering is a time-consuming task as it requires each frame to be copied from the GPU into main memory and send over the network to other processes. In order to reduce the size of the transferred frames compression is used. However, this tends to occur after the frames have been copied into main memory. Lipinski et al. [96] leverage OpenGL and CUDA in order to compress raw images on the GPU prior to transferring the data to main memory and sending it over the network. Eilemann et al. [37] developed a parallel OpenGL-based middleware named Equalizer, which supports scalable display environments. It provides a large set of features to support a wide variety of research and industry applications. OmegaLib [39] is based on Equalizer and provides tools to develop immersive 2D and 3D applications for systems ranging from large display walls to CAVEs. It supports dynamic reconfiguration of the display environment in order to interactively allocate 2D and 3D workspaces. However, OmegaLib focuses on co-located collaboration and does not support real-time communication between multiple sites.

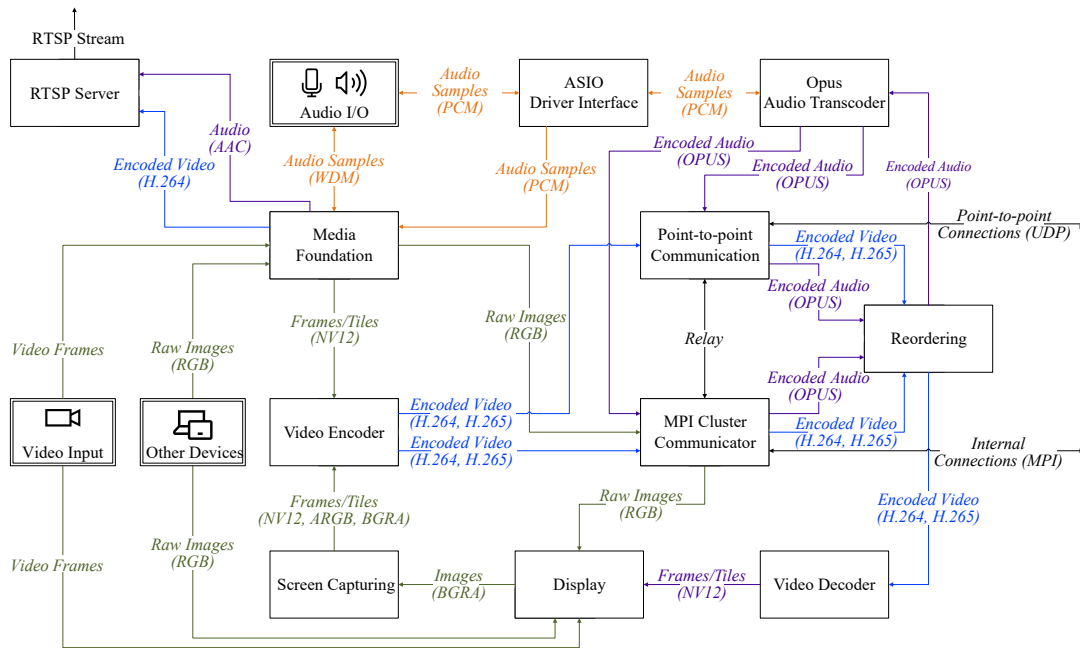
Different solutions for interactive high-resolution streaming of visualisation content have been developed in recent years. Biedert et al. [15] developed a streaming solution using video compression to achieve high frame rates. This solution is similar to the system, discussed in this chapter, for screen sharing but it requires the same hardware configuration for the source and destination display, whereas the presented system is able to work with different configurations. Additionally, they synchronise all encoded video streams on the server side and the client side to ensure a smooth playback. This synchronisation introduces additional latency that increases with each additional node and it can lead to dropped frames on server nodes, if they cannot keep up with the frame rate of the other server nodes. Marrinan et al. [104] showed a system that allows streaming high-resolution content, without using any compression, in real-time. Their server is a combination of multiple TCP socket servers, i.e. they launch a TCP server for each node that contains a part of the final image. This allows for parallel processing of rendering and transmitting a frame to one or more clients. Each of the  $N$  clients will connect to  $\frac{M}{N}$  of the  $M$  servers and receive frames, which will be redistributed to match the layout of the tiled display. Although the system is able to reach high frame rates without dropping frames it needs a connection with a very high bandwidth to do so. Furthermore, in contrast to the presented system, neither approach supports additional audio and video streams.



**Figure 4.3** — An example of the screen sharing scenario as displayed on the tiled display on site A: site B is sharing the content of their large high-resolution display (seen on the left). As the aspect ratio is wider on site A, the 4K video stream of the presenter (seen on the right) can be shown beside the shared visualisation. Bidirectional audio streams allow users to talk with each other while investigating the visualisation shared. © EG 2020 [46].

## 4.1 System Architecture

The architecture and implementation of our system is defined by three main goals: first and foremost, the software should be based on moving pixels around while leveraging any hardware support available, in order to see whether the current graphics and network hardware would allow application-agnostic interactivity to be achieved even on large high-resolution displays. Second, the system should support several real-world scenarios, namely using the system for on-site and remote presentations in the way a standard projector would work (cf. Figure 4.2), sharing the full-resolution tiled display screen between remote sites in an application-agnostic way (Figure 4.3) and combining both scenarios with video and audio streams for a video conference setup. The target resolution for the video streams was 4K in order to match the visual fidelity available through the large high-resolution displays. Third, the system should run in different kinds of tiled display environments, namely the ones using separate cluster nodes to drive the tiles as well as the ones using as many GPUs as possible in a single machine to drive the whole display with one computer. In the original usage scenario shown



**Figure 4.4** – Overview of the components of the system and data flow between them. The diagram covers only components on one site, the other site uses the same setup, potentially configured differently. The orange connections represent raw audio samples whereas encoded audio samples are represented by purple connections. Raw video frames are shown in green and encoded video frames are blue. The connections noted in black represent any of the above, i.e. data-agnostic communication links. © EG 2020 [46].

in Figure 4.1, the tiled display at site A has a resolution of  $10800 \times 4096$  pixels, which is composed from ten projectors (five for each stereo channel) driven by a cluster of 20 nodes. The display at site B comprises  $5224 \times 2160$  pixels from two projectors driven by a single node. At both sites a dedicated computer, the streaming node, handles all the video and audio devices and also has the required internet visibility to serve streams to the other site and to RTSP clients. As detailed in section 4.2, these tasks could also be handled by other machines, but it greatly facilitates the physical setup and the configuration of the network.

The next sections describe – by following audio and video samples from obtaining the original input to audio and video rendering – how the system was designed to achieve the aforementioned goals.

### 4.1.1 Input Streams

The goal of streaming live videos at high resolutions and exposing the tiled displays as standard projection devices to end users bringing their own laptops mandates the use of hardware frame grabbers. Such devices like BlackMagic Design's DeckLink family or the Unigraf UFG family – the two device families are used in the conference scenario shown in Figure 4.2 – have significantly fallen in price over the recent years and are a simple solution for achieving this goal. While vendors of such extension cards typically include their own software development kit for interacting with the cards, they also include drivers to support DirectShow and/or Windows Media Foundation on Microsoft Windows platforms. As using the latter enables writing mostly hardware-agnostic software when it comes to obtaining video signals, Windows Media Foundation is the base of the system. As depicted in Figure 4.4, Media Foundation plays a central role when it comes to processing video and audio samples. It is the sole source of external video signals from cameras and frame grabbers for capturing the screen of a laptop. However, there are additional sources outside the scope of Media Foundation, which are audio samples from the low-latency ASIO (Audio Stream Input/Output) API [151], video samples from the Desktop Duplication API [105], which is used to perform the application-agnostic transmission of the full-resolution, tiled desktop and audio and video streams originating from the remote site (see Figure 4.3).

### 4.1.2 Encoding

Video and audio streams transmitted over networks need to be encoded to match the available bandwidth of the network. While Media Foundation provides a variety of out-of-the-box encoders, many of them are software-based or can use the built-in hardware encoders of modern GPUs only if certain preconditions are met. The evaluation of the Real-Time Streaming Protocol (RTSP) in combination with the built-in encoders of Media Foundation for the system showed that the latency introduced by this approach was unacceptable for the video conference part of the software. In order to overcome these issues, the point-to-point communication between the two tiled display sites uses Opus [72] for audio en-/decoding and the GPU hardware en-/decoder for video, available on all reasonably new graphics cards. Both, the video and the audio path, are implemented outside Media Foundation as wrapping this functionality in a Media Foundation transform would have required additional coding effort while adding additional latency at runtime. Nevertheless, the original RTSP path is still part of the system and allows external users without direct access to the tiled display installations to watch presentations, using the system, with a standard video player.

### 4.1.3 Network Transfer

The network technologies used in the system fall into two categories: site-local communication between computers used to drive a single tiled display, and internet communication to transmit audio and video data between the sites. The former is implemented using the Message Passing Interface (MPI), which is the de-facto standard for communication in HPC clusters. Besides providing a portable API, MPI adds the additional advantage of having specialised implementations for high-speed network technologies like InfiniBand (IB) that talk almost directly to the hardware. For instance, InfiniBand implementations of MPI use IB verbs bypassing the whole TCP/IP stack (in contrast to IP over IB), thus greatly reducing network latency and increasing bandwidth. On systems where no such implementation is available, MPI can be used on top of TCP/IP and standard Ethernet connections. Ethernet and IP are also the means for connecting the two remote sites. As already mentioned previously, using an out-of-the-box RTSP implementation proved to add too much latency for video conferences, therefore the system uses a UDP-based protocol, which closely resembles the packet format of the Real-Time Transport Protocol (RTP), but strips off the session management part of RTSP.

### 4.1.4 Rendering

The use of technologies like Windows Media Foundation and the Desktop Duplication API strongly suggests using Direct3D as rendering API. The main reason for that is that since the introduction of the Windows Display Driver Model (WDDM) in Windows Vista, most of the graphics in Windows are based on the DirectX Graphics Infrastructure (DXGI). This design implies that the representation of hardware-accelerated video samples in Media Foundation, the output of the Desktop Duplication API and 2D textures in Direct3D are all the same and can be seamlessly shared between the APIs. Furthermore, Direct3D has a long history of being very explicit when it comes to exposing the display topology to the programmer. The fact that it allows addressing each GPU and their respective outputs explicitly is a valuable feature when implementing software for tiled displays that typically include multiple GPU and multiple display scenarios. The actual implementation of the rendering is straightforward: all video samples arrive in the form of a Direct3D texture, regardless of their source, which is rendered on a screen-aligned quad at the right position on the screen. For encoded video streams decoding is performed, using a GPU hardware decoder, in order to get a Direct3D texture containing the frame.

Audio rendering can be handled by the streaming audio renderer of Windows Media Foundation or using ASIO, which has been designed for minimal latency. Furthermore, depending on the underlying hardware, ASIO allows a large number of audio channels



to be addressed individually, which can be used for positional audio in an immersive display installation.

### 4.1.5 Configuration

As previously outlined, the goal was to support a variety of different usage scenarios, which differ in the inputs and outputs they use as well as in the display layouts. In order to manage these different scenarios, a declarative approach using XML is used. All computers involved in a scenario – display nodes connected to the output as well as dedicated streaming nodes – are configured in the same file and identified by pertinent strings such as the host name, an IP address or one of the MAC addresses of the computer. Likewise, input and output devices are identified by their hardware IDs, and network streams by their URL. All of these resources can be connected in a graph-like manner, which allows the system to build the corresponding in-memory representation at startup. Furthermore, the configuration file also allows the configuration of most components, e.g. the settings for the encoder and decoder or specific settings for the sources.

## 4.2 Implementation Details

Given the use of Direct3D 11, a variety of native Windows APIs, multiple SDKs from NVIDIA, AMD and Intel as well as third-party and open source libraries, which are natively written in C/C++, the system is written in C++ as well such that it can make direct use of these APIs. The NVIDIA SDKs in question are the Video Codec SDK (version 11.1) [111] for en-/decoding H.264/H.265 videos on the GPU, and the NVAPI for addressing the proprietary framelock feature of Quadro GPUs. The Video Codec SDK contains two hardware acceleration interfaces: NVENCODE (NVENC) for video encode acceleration and NVDECODE (NVDEC) for video decode acceleration. The Advanced Media Framework SDK (version 1.4.21.0) from AMD [7] and the Intel oneAPI Video Processing Library (oneVPL) (version 2021.6.0) [71] perform the same en-/decoding tasks for AMD and Intel GPUs respectively. x264 [163] and x265 [164] can be used for encoding on the CPU in case the system does not contain a GPU to do the encoding. The most relevant third-party libraries are the ASIO SDK from Steinberg, which allows accessing low-latency audio devices, the Opus codec for encoding and decoding the audio streams in the video conference scenario and the live555 [99] streaming library to implement the RTSP server. In order to process all data in real time, the implementation is as efficient as possible. For instance, zero-copy implementations on the CPU are used whenever possible and memory allocations are avoided by using pools of frequently used objects. Naturally, moving data between CPU and GPU is avoided, instead texture handles are passed around. GPU-to-GPU copies are used only if it cannot be avoided.

The following sections describe how all of this is integrated into the overall system depicted in Figure 4.4 and the interplay between the components of the system.

### 4.2.1 Media Foundation

Windows Media Foundation is at the core of video and audio processing. It is the only source of external video data (cameras and frame grabber cards) and can be used as audio source. Furthermore, it provides a variety of format and colour conversions as well as resampling transforms. At the heart of Windows Media Foundation is a data flow graph (topology) which connects media sources, sinks and optional transforms. As the software starts, the user-defined application scenario is translated into such a topology containing all necessary input sources and output sinks. Converters and resamplers are then mostly inserted automatically by Media Foundation as it resolved the final topology. Unfortunately, not all conversions required for the software were available and ready to use. For instance, Media Foundation does not understand the video format used by Blackmagic Design's capturing cards in 4K mode. Therefore, a special transform node was developed to make such devices usable. The same holds for interfacing with ASIO (see subsection 4.2.2). The majority of the sinks used extract data from the Media Foundation topology are based on so-called sample grabbers. Sample grabbers provide an easy-to-use way to extract raw and encoded audio and video samples without having to implement the complicated control flow of generic sinks, which require the implementer to request samples at regular intervals and keeping track of these requests as the samples flow in from the preceding nodes. From a programmer's point of view, a sample grabber implements a push-based data flow where samples arrive as they are ready. The most important sample grabbers that were implemented, extract video samples for encoding using the GPU hardware encoder for the video conference stream and the audio and video sinks from the RTSP server, which receive encoded H.264 and AAC samples.

### 4.2.2 ASIO

Steinberg developed ASIO [151] specifically with low-latency in mind. This constitutes a more than welcome side effect of the fact that the system had to support ASIO beside Media Foundation as the audio technology at one of the test sites is based on ASIO. Provided compatible hardware is used, ASIO additionally allows a large number of output channels to be addressed individually, which is beneficial for large high-resolution display setups featuring multiple loudspeakers for positional audio. Technically, ASIO devices are addressed by copying raw audio samples to and from its input and output buffers at regular time intervals. In order to feed audio samples from ASIO devices into the Media Foundation path, a live source that converts ASIO samples to Media Foundation samples was implemented. However, this adds latency

due to another ring buffer decoupling the ASIO sample generation rate from the Media Foundation consumption rate. Therefore, a short-cut, directly from the ASIO source to the Opus encoder and in turn to the UDP stream, is provided. This path is used for the audio channel of the video conferences and includes all necessary conversions like ensuring that input to Opus must be interleaved PCM16 samples. Likewise, in the opposite direction, samples being received from the network can be directly copied into the ASIO playback buffers from the Opus decoder. If the audio stream and the corresponding video stream are processed on different nodes, playback can potentially become unsynchronised. Therefore, playback of incoming audio samples can be delayed by a constant factor in order to manually synchronise audio and video.

### 4.2.3 Screen Capturing

Starting with Windows 8, the Desktop Duplication API provides fast access to the full desktop image as a texture. Using this API, applications can receive periodical updates from each of the desktops attached to the system as BGRA texture, requiring a separate desktop duplication session for each display. In contrast to previous APIs, the data stay on the GPU allowing the application to decide how to proceed. As the desktop textures are directly passed on to the hardware encoder in the screen sharing scenario, each of the duplication sessions needs a separate encoder session as well. Since a single desktop texture might exceed the maximum supported resolution of the hardware encoder, e.g. when using technologies like NVIDIA's MOSAIC or recent 8K displays, different options are available: bilinear downscaling and tiling. Tiling divides the texture into multiple smaller textures that are then encoded individually by a separate encoder session. GTX, RTX and Titan cards from NVIDIA are restricted to only two encoder sessions in parallel, therefore Quadro type cards are required in case that more than two tiles are encoded, as they do not have an upper limit for parallel sessions. Neither AMD nor Intel limit the number of parallel encoder sessions for their respective GPUs. It is also possible to use one session for multiple tiles, at the cost of additional latency and bandwidth as every frame has to be encoded as a key frame. The first request to the desktop duplication API, in order to get the first frame, is synchronised across all displays. Further requests are not synchronised as it is possible that the Desktop Duplication API misses a frame on one node but not on the others. Since those occasionally missed frames are not noticeable, a full synchronisation is not needed as it would hamper the other nodes as well.

The Desktop Duplication API does not handle display rotations in the way it is apparent to the user in front of the hardware. The content of the texture is rotated, therefore directly rendering the output of displays having their output rotated does not behave as expected. The texture therefore needs to be rotated appropriately, in 90° steps, before rendering, which can be done either before the frame is encoded on the source or after



it is decoded on the client side. As there is already a compute shader that performs the necessary conversion from the BGRA to the ARGB colour format as well as the optional downscaling or tiling, it also performs the rotation. Technically, the rotation is implemented without actually moving pixels, but by rotating the texture coordinates. Frames are encoded and decoded in exactly the same way as for any other stream (see subsection 4.2.4 and subsection 4.2.8).

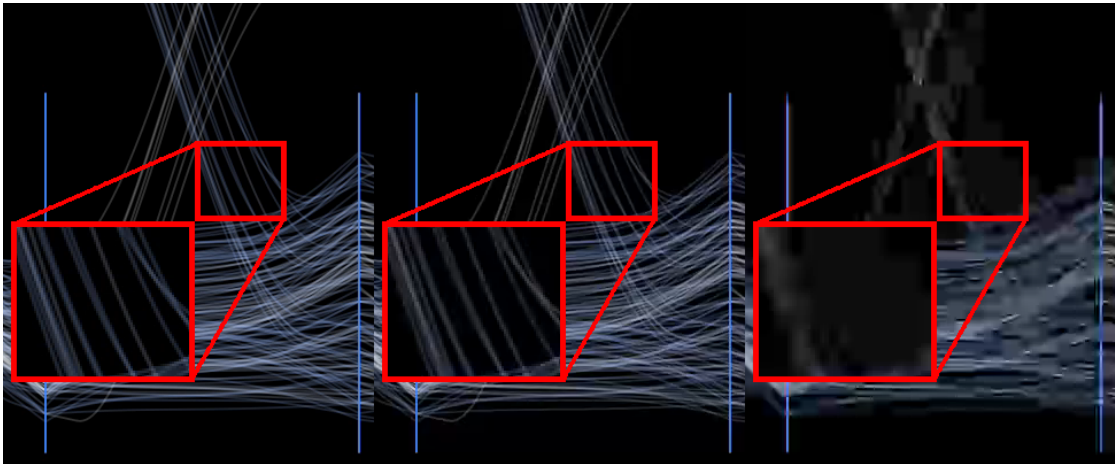
The system allows different modes for transmitting the captured desktop streams. The first one uses a dedicated node receiving all encoded streams, either via MPI or UDP, and forwarding them to the display nodes. The second one allows for a direct transmission to the display nodes similar to the system proposed by Biedert et al. [15]. It is also possible to combine the two modes.

#### 4.2.4 Encoding

Audio and video streams need to be encoded in real time before they are transmitted to a remote site as all sources are live sources. Processing data from the stream sources is implemented as a lock-free asynchronous approach using ring buffers. Encoding (and decoding) happens asynchronously in separate threads or in hardware. For frame sources such as sample grabbers or network threads, encoding is a fire-and-forget operation.

For the audio streams the Opus codec [72] is used, which has been derived from the open-source format CELT and Skype's SILK codec. Opus is an open standard that has been specifically designed for real-time communication over the internet. Its use is straightforward as the programmer just copies the interleaved PCM16 samples into an input buffer and receives the encoded byte stream.

Video streams are encoded using either the H.264 codec or the H.265 codec. For the encoding, each of the aforementioned SDKs (Advanced Media Framework SDK, Intel oneAPI Video Processing Library and Video Codec SDK) can be used. Each provides direct access to the dedicated hardware encoder and allows encoding of the frames without piping them through main memory as they are already on the GPU, which is an important advantage when combined with the speed of the encoder. Each encoder session uses a ring buffer which contains the input data and the output data per element. This ring buffer needs to be initialised before the session can be used and there are strict requirements for the format of the input data, depending on the SDK used. For NVENC it needs to be either CUDA memory on the GPU or a Direct3D 2D interop texture, which provides access to the same underlying memory on the GPU through CUDA and Direct3D. For the AMD and Intel SDKs the input needs to be a Direct3D 2D interop texture. Using interop textures as the input for the encoder in screen capturing (see subsection 4.2.3) avoids an additional copy after the colour conversion and scaling of the captured frame. While all SDKs support a wide range of colour formats the



**Figure 4.5** — The resulting decoded image when using the three different encoder settings, from left to right: high, medium and low. The red box shows the lines at a 200% zoom level. The high setting, with a QP value of 11, preserves the image quality of the original image at the cost of an increased size of the encoded image. The medium setting, with a QP value of 31, comes close to the original image quality but, as can be partially seen in the zoomed section, already introduces artefacts for fine structures. The low setting, with a QP value of 51, fails to preserve any fine details but requires much less bandwidth.

compute shader (see subsection 4.2.3) only supports conversions to NV12, ARGB and BGRA (which is only supported by AMD and does not require any conversion). This covers the most frequently used colour formats used for encoding. For NVENC there is an additional requirement: the stride, i.e. the width of a line of pixels in bytes, needs to be a multiple of 512.

Encoding happens in-place and asynchronously for all SDKs, while NVENC also offers a synchronous encoding mode. Any data that needs to be encoded is copied into the next free element of the ring buffer, with the encoder signalling an event, in the case of asynchronous encoding, for this element once it has been encoded. Once the frame is encoded, it is downloaded, sliced to fit the MTU size of the underlying network and is sent to the consumers via point-to-point connections (cf. subsection 4.2.7).

### 4.2.5 Encoding Settings

All SDKs provide a wide range of configuration options for the encoders. The configuration file allows changing most of the parameters for each stream individually. For camera streams, a maximum bit rate is used and the encoder adapts the quality para-

meter of the frames to stay within the allowed bandwidth. For the streams of external content, i.e. the laptop of a presenter, a constant quantisation parameter (QP) is used in order to keep the quality as high as possible and the content of the presented slides readable. The constant quantisation parameter can be set to integral values between 51 and 1 with 51 resulting in the lowest quality and 1 in the best quality – basically turning off any compression. For screen sharing (see subsection 4.2.3), three pre-defined quality settings are provided: low, medium and high, see Figure 4.5. All settings use a constant quantisation parameter, so the bit rate of the encoder is not limited. Furthermore, if possible, an infinite GOP length with a regular intra refresh is used as an error resilience measure. If this is not used, the GOP length is limited and therefore key frames are inserted regularly. The low quality setting reduces the necessary bandwidth at the cost of a visibly reduced image quality by using a constant quantisation parameter of 51. Streaming the content of a 44 Megapixel display at 60 frames per second, requires at most 100 Mb/s using this setting. The medium quality, with a QP value of 30, offers a typical compromise between bandwidth and quality, which is sufficient for most visualisation applications. The high quality setting has large bandwidth requirements, because the constant quantisation parameter is set to 11. Since the quality setting can be set for each captured display individually, it is possible to adapt the overall quality by assigning different settings to the displays, which would allow for scenarios that use higher quality streams for the centre of the display than for the borders.

#### 4.2.6 MPI Cluster Communicator

Communication between nodes at one site is implemented using MPI. While the advantages of this approach – mainly direct access to the low-latency InfiniBand network – have already been outlined, this choice also comes with a drawback. Most MPI implementations, although offering non-blocking calls, require all API calls to be made from a single thread. However, the producers (Windows Media Foundation, NVENC and network streams) are all asynchronous, mandating all data being funnelled through another message queue before being transmitted to local nodes. Furthermore, as the communication patterns in the local cluster depend on the user-provided configuration of streams, the implementation of that pattern must ensure that user input cannot result in deadlocks due to mutually pending unicast operations. Therefore, the number of communication calls is minimised by coalescing all data packets queued during rendering a frame into one chunk for each stream type and send it in one packet via MPI, which has the added benefit of increasing the network throughput. Additionally, all nodes send these chunks once per frame using a non-blocking operation to all other nodes that subscribed to the respective streams. Then, non-blocking receive calls are made, after which all send and receive operations are awaited to safely recycle the memory used for the transfer.

Based on the user-supplied configuration, each node in the local cluster is assigned one role or a combination of two roles: stream providers and stream receiver. Stream providers are nodes that publish audio or video data originating from them – including samples from sharing their desktop – or that relay such data from a network source. If a node shares its desktop, it is assigned the stream provider role as well. Stream receivers are nodes that show at least one video stream (or play an audio stream). As the software starts, all local nodes exchange their roles and the streams they are interested in among each other, which allows for the optimisation of the per-frame communication so that only necessary point-to-point transfers are initiated.

### 4.2.7 Point-to-point Communication

Communication between sites is carried out using UDP to achieve the best possible saturation of the network and to avoid latency introduced by TCP. Technically, all send and receive operations are processed asynchronously using an I/O completion port (IOCP) [106]. An IOCP is a queue for completion events of asynchronous I/O operations managed by the operating system. A thread pool sized according to the native parallelism of the underlying hardware is used to process the completion events. This way, all internet communication is implemented in one place, which can control the life time of the memory used for data transfer. This memory is pooled in chunks of the largest packet the system has seen, which avoids costly heap allocations in every frame.

The UDP datagrams containing frames might arrive in an incorrect order on the receiver side – or not at all –, which is problematic for most decoders. Therefore, a ring buffer on the receiver side is used to reorder packets based on a sequence number sent with each frame before passing the data to the decoders. The size of this buffer is configurable as it adds latency or causes discontinuities if packets arrive that are too far in the future, i.e. that would make the buffer overflow. In such a case, the new packet defines the new reference time and all previously received packets are dropped in the same way as any other packet that is behind the reference time. Additionally, if after a user specified time no packet has been dequeued all missing packets are dropped. This is done to give the user more control on how to handle missing packets. Usually the time is set to the duration of three packets, e.g. for a 60fps video stream the maximum time would be 48 ms.

### 4.2.8 Decoding

The hardware video decoder is initialised on the fly since it needs the settings from the encoder, which clients receive every time when connecting to a remote stream. It works, in a similar fashion to the encoder, on its own internal ring buffer on the

GPU. The encoded frames are queued to the decoder. For NVDEC this triggers an asynchronous chain of callbacks. The other SDKs (AMD and Intel) require a polling mechanism, which can be performed by a separate thread, to check if the frame is fully decoded. The decoded frame is then queued into a separate ring buffer (display queue) used to hold the frames until they are displayed. This ring buffer uses Direct3D interop textures again in a way that the decoded frame can be displayed directly. For the AMD SDK the display queue is the internal list of output surfaces, owned by the decoder, which are Direct3D interop textures. As long as a surface is marked, i.e., is queued in the display queue, the decoder does not use it, therefore no copy operation is required. Both the NVIDIA and Intel SDK require a copy operation from the output surface to the display queue.

The Opus decoder works similarly to the Opus encoder and provides interleaved PCM16 samples for chunks of the encoded byte stream. They need to be deinterleaved and changed to the correct PCM format before being queued for playback by the ASIO device, which happens in an asynchronous continuation function.

### 4.2.9 Display

Rendering using Direct3D is more or less straightforward drawing the video samples as textured quads. All scaling and positioning transformations are performed on the fly on the GPU, which includes clipping the video to match the segment of the tiled display the respective node is responsible for. Whenever possible, data already resident on the GPU is used for rendering. Furthermore, typical video texture formats like UYVY, YUYV or NV12 are converted to RGB(A) directly in the pixel shader, avoiding transfers between GPU and main memory and reducing the bandwidth for transfers at the same time.

If a video stream is displayed across multiple nodes, each node must show the same frame at the same time in order to avoid tearing artefacts. There are two different techniques implemented to synchronise the playback. The first is a standard MPI barrier synchronising all nodes before the buffer swap of the next frame. Since all nodes receive the same frames and they are ordered based on the timestamp the next frame is the same for all nodes. This solution is portable, but MPI barriers are comparably expensive operations and still cannot ensure that the buffer swap happens at exactly the same time. When supported by the hardware, synchronisation based on NVIDIA's Quadro Sync technology [110] is used. Quadro Sync uses an proprietary protocol in the graphics driver and a separate Ethernet network to ensure that all displays swap buffers at the same time. This makes it the preferred way to synchronise the output over multiple display nodes as it does not interfere with MPI or any other component of the software.

### 4.2.10 RTSP

For scenarios in which a person is giving a talk at one site which is transmitted to another one, it is desirable to enable external viewers to connect to the stream. However, these clients have no large high-resolution display available and the goal was to allow users to be able to connect with standard technologies. Therefore, an RTSP server based on live555 was integrated. Each node within the cluster can theoretically be configured to be the RTSP server. Typically, this role is configured on the dedicated streaming node (cf. Figure 4.1), which has direct hardware access to the video and audio sources. Encoded audio and video data for the RTSP stream as well as synchronisation of audio and video samples are provided by Windows Media Foundation.

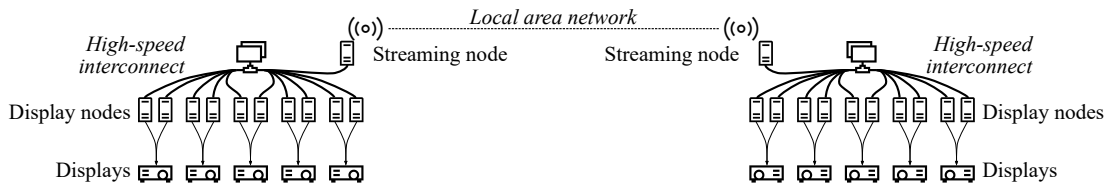
If the same streams are sent to a remote tiled display and via RTSP, a so-called T node is added to the Media Foundation topology, which allows for the duplication of incoming samples to both processing paths. Video samples are resampled to full HD resolution for the RTSP path and then encoded using the H.264 encoder provided by Media Foundation. Likewise, Media Foundation's AAC codec is used for the audio samples. At the end of the pipeline, all encoded samples are extracted from the topology using a sample grabber in a bridge module that was written to connect Media Foundation to live555. This module decouples the pull model used by live555 and the aforementioned push property of the sample grabber by means of a ring buffer. Furthermore, the RTP stream requests individual Network Abstract Layer Units (NALUs) whereas the H.264 encoder typically produces multiple NALUs per sample. The bridge module therefore splits each sample into its individual NALUs and forwards these separately to the stream as requested. The increased processing overhead, and the fact that the current Media Foundation-based implementation uses software encoders and video players which typically perform some buffering to compensate intermittent network problems eventually lead to a noticeable latency of the RTSP stream of around three seconds compared to the point-to-point stream.

## 4.3 Test Setup

The same hardware setup was used for the evaluation of the system, described in this chapter, as well as the optimisations discussed in chapter 5 and chapter 6. In addition, the same molecular dynamic simulations were used in order to compare the different approaches to each other and to the baseline system. The following paragraphs go into more details regarding the hardware used as well as the simulations.

**Hardware Details** The stereo capable tiled display, with a resolution of  $10800 \times 4096$ , used for the tests is driven by 20 display nodes that are each equipped with an NVIDIA Quadro M6000, two Xeon E5-2640v3 CPUs and 256 GB of RAM. Each is responsible for



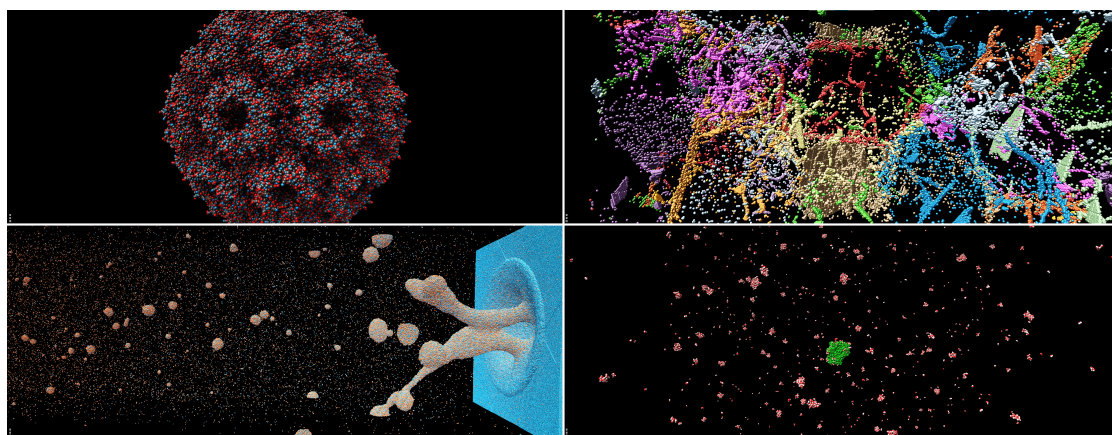


**Figure 4.6** — The Hardware setup that was used for all tests that measured the throughput and latency using the large high-resolution display. The two streaming nodes are connected using a 10-Gbit/s-Ethernet network adapter while the display nodes are part of the low-latency InfiniBand network. © IEEE 2020 [44].

a tile of  $1200 \times 4096$  pixels of the overall screen, with 10 nodes per stereo channel. Two streaming nodes, which are part of the low-latency InfiniBand network that connects the display nodes, provide the connection between the server and the client side over a 10 Gb network adapter. Both are equipped with an Intel Core i7-6850K, 64 GB of RAM and a GeForce GTX 1060. All machines, the display nodes and the two streaming nodes, run on Windows Server 2016. Figure 4.6 shows a schematic overview of the hardware setup that was used for the evaluation. The left half shows the server side, while the right half shows the client side. They are connected to each other via the 10 Gb network adapters of the streaming nodes.

Since the NVIDIA Quadro M6000 is unable to decode H.265 in hardware all tests are done using H.264 codec. Additionally, two machines were used to evaluate the difference between H.264 and H.265 regarding latency and throughput. The machine performing the encoding is equipped with an NVIDIA GeForce GTX 960, a AMD Ryzen 7 1700X CPU and 32 GB of RAM and captures a display with the resolution of  $2560 \times 1440$ . The machine performing the decoding is equipped with an NVIDIA Quadro RTX 8000, two Intel Xeon Gold 6128 CPUs and 192 GB of RAM.

**Visualisations** The four visualisations in Figure 4.7 are all rendered using MegaMol, described in subsection 2.5.2 and cover different changes, on a frame-to-frame basis, in order to evaluate the impact on latency and bandwidth of the system and the optimisations, discussed in chapter 5 and chapter 6. The first visualisation, *mdao* has very small changes from frame to frame, which benefits the compression as the intra and inter prediction. The second, *crystal*, has small changes between successive frames with rare larger changes, while the third, *laser*, has big changes for the first third of the simulation and then also very small changes from frame-to-frame. The fourth, *solvent* has a combination of small and larger changes on a frame-to-frame basis due to the slow moving protein in the centre of the image and the fast moving water molecules.



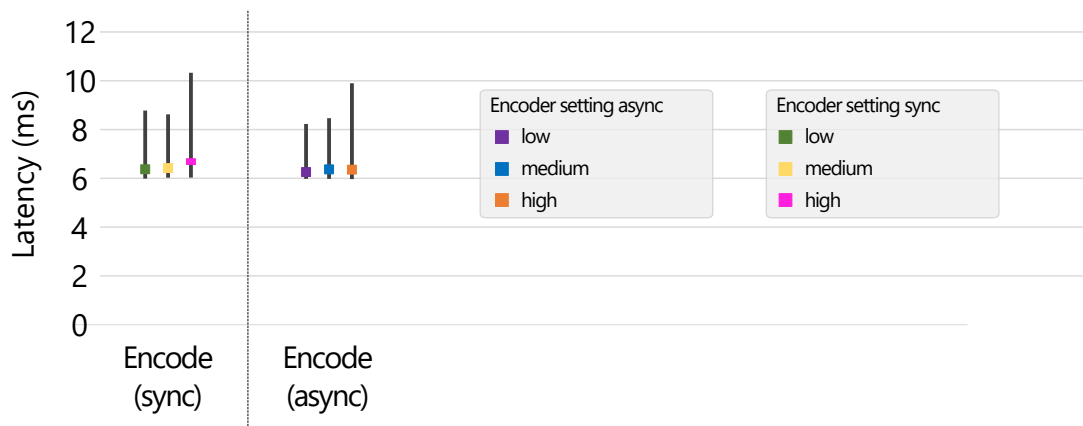
**Figure 4.7** — The four visualisations used to evaluate the latency and throughput of the system. On the top left the *mdao* represents visualisations with little change between successive frames, while the *crystal* (top right) and *laser* (bottom left) represent visualisations with a mixture of small and occasional bigger changes on a frame-to-frame basis. Finally the *solvent* (bottom right) dataset represents visualisations with bigger changes between successive frames. © IEEE 2021 [43].

**Method** In order to measure the latency on-site and across sites on different nodes the algorithm by Cristian [31] was used to synchronise the clocks of the nodes to an external time source. Then for each major step in the pipeline of the system, e.g. encoding and decoding, the latency is measured by taking the duration of each individual operation and adding it to a sliding window of size 2000. From this window the minimum, maximum, average and median duration as well as the median absolute deviation is computed for each operation. For the throughput measurement the size in bit of the incoming data is added up and after one second the Mb/s for this second is computed. The same is done for the outgoing data.

## 4.4 Results and Discussion

A quantitative evaluation of the system was performed by measuring the latency and throughput required to share the content of a tiled display in a local area network setting as described in section 4.3. This local setting provides a controlled network environment as the content is transmitted from the right to the left stereo channel of the tiled display. The throughput and latency was measured using four different molecular dynamics visualisations, also described in section 4.3.

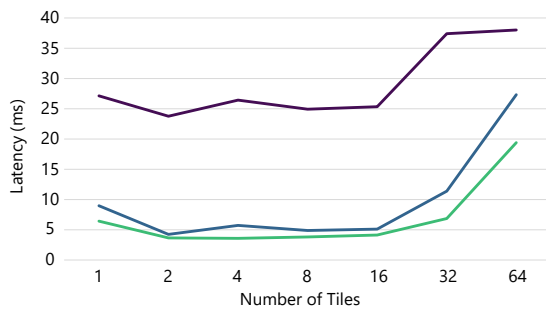




**Figure 4.8** — The median, maximum and average latency, in milliseconds, for the encoding using the *crystal* dataset and the synchronous (sync) and asynchronous (async) encoding modes. The minimum and maximum is indicated by the grey lines, while the coloured block shows the median value. As can be seen, there is barely a difference between the synchronous and asynchronous encoding, with the median latency between 6 and 7 ms for all three settings, low, medium and high. The maximum latency increases from around 8 ms for the low and medium setting to around 10 ms for the high setting. The minimum latency is around 6 ms for all three settings.

**Test Scenarios** The first test was done to evaluate the encoding mode of the NVENC, since the SDK offers both, synchronised and asynchronous encoding. The second test looked at the performance impact of downscaling and tiling, while the third test evaluated the difference between H.264 and H.265 using the *solvent* dataset. After performing these preliminary tests two scenarios were tested. In the first test scenario the tiled display was shared while showing each of the four molecular dynamics simulations. This scenario provides a controlled network environment and it does not involve any video scaling as it uses the left and the right stereo channel of the tiled display. For the second scenario the video conference scenario was tested by transmitting full HD presentation slides, bidirectional 4K video as well as audio from site A to site B.

**Test Results** Figure 4.8 shows the time in milliseconds required for the encoding using the asynchronous mode and the synchronous mode for the *crystal* dataset. The values represent the worst case measured across all ten nodes, while the other nodes only differ significantly for the maximum latency, with a difference of  $\pm 1$  ms. As can be seen, there is no difference between the two modes, therefore the remaining tests use the asynchronous mode in order to avoid blocking the screen capturing. The minimum latency for all three settings, low, medium and high, is around 6 ms while the median



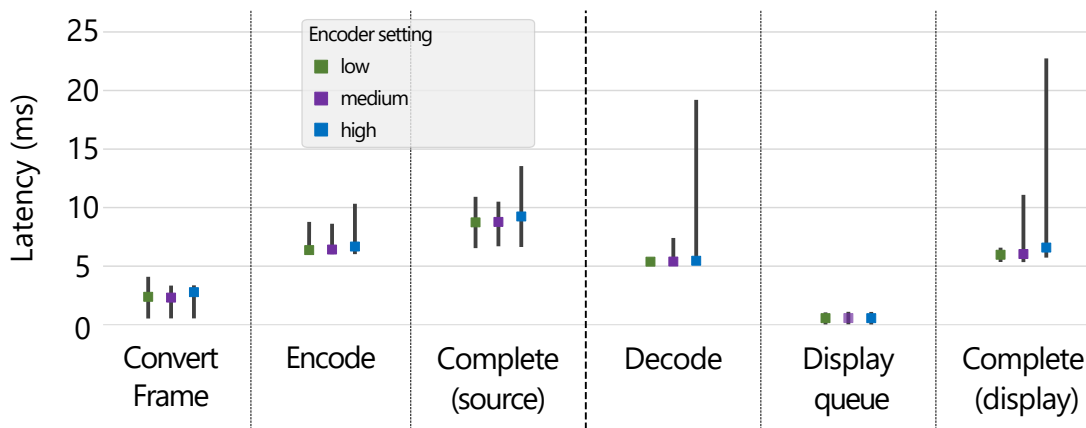
**Figure 4.9** — The maximum (■), median (■) and minimum (■) latency, measured in milliseconds, from capturing a frame to sending it for 1, 2, 4, 8, 16, 32 and 64 tiles for the *laser* dataset. Between two and 16 tiles the minimum and median latency decreases from 6.4 and 8.9 ms to roughly 3.5 and 4.0 ms, while the maximum latency stays at roughly 25 ms. Using more than 16 tiles increases the latencies.

latency is between 6 and 7 ms and the maximum latency is 8, 8 and 10 ms respectively.

The second preliminary test looked at the impact of the downscaling and the tiling, both of which can be used to decrease the resolution in order to stay below the maximum resolution supported by the encoder. While the downscaling reduces the resolution of a single frame, the tiling divides the image into multiple smaller frames. Using the downscaling the time required to convert the frame (cf. subsection 4.2.3) increases slightly by approximately 1 ms when scaling a frame from  $1200 \times 4096$  to  $600 \times 2048$ . Tiling, on the other hand, reduces the time it takes to encode a complete frame, i.e. all tiles, significantly. Figure 4.9 shows the impact on the latency from capturing a full frame to sending it for 1, 2, 4, 8, 16, 32 and 64 tiles using the high encoder setting and the *laser* dataset. The results are similar for the other two encoder settings, low and medium. As can be seen, using more than one tile first decreases the latency until the number of tiles reaches 16. After this the latency begins to increase again and for more than 32 tiles the latency is higher than for one tile. The decoding latency is largely unaffected by this and stays roughly the same until it also increases at more than 16 tiles. For two, four and eight tiles the minimum and median latency is roughly half of the respective latencies for one tile, with around 3.5 ms and 4.2 ms respectively. The maximum latency is largely unaffected until it also increases for more than 32 tiles.

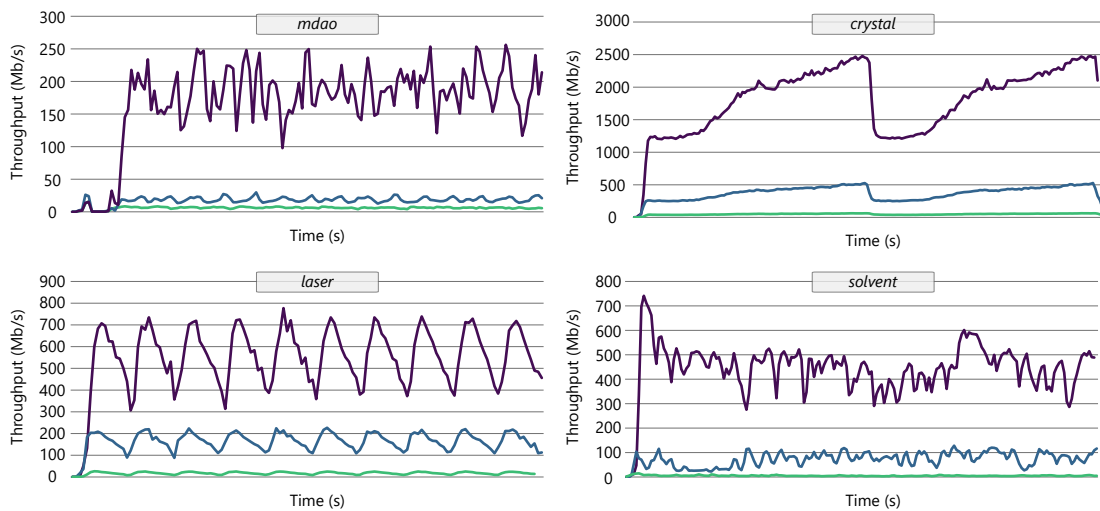
The third preliminary test looked at the difference between H.264 and H.265 for a display with the resolution of  $2560 \times 1440$ , since the Quadro M6000 GPUs that drive the large display do not support decoding of H.265. Using the H.265 codec for the *solvent* dataset reduces the required bandwidth from at most 3.7 Mb/s to at most 2.5 Mb/s for the low settings, from at most 38.9 Mb/s to 35.1 Mb/s for the medium setting and from 148.3 Mb/s to 134.5 Mb/s for the high setting. The minimum and median latency on the other hand increase by 4 ms for each of the three encoder settings, while the maximum increases by 3 ms.

Figure 4.10 shows the latencies, in milliseconds, for the major steps of the controlled,



**Figure 4.10** — The measured latencies, in milliseconds, for the major steps of the encoding and decoding pipeline using the *crystal* dataset and the three different encoder settings. The underlying values are the worst case latencies measured across all nodes, with the variance of other nodes below 1 ms. The plot shows the median values (coloured block) as well as the minimum and maximum latencies, indicated by the grey lines. *Complete (Source)* shows the complete duration from capturing the frame to sending it. Likewise, *Complete (Display)* depicts the complete duration from receiving the encoded frame to displaying it. The encoding and decoding have the biggest impact on the overall latency with minor differences between the settings for the encoding and a major difference for the decoding using the high setting. For the decoding the difference between the settings is greater, especially for the high setting.

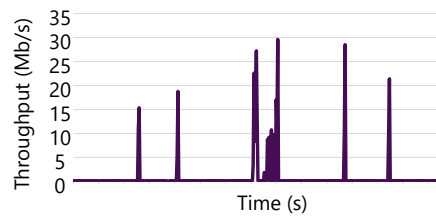
local screen sharing scenario using the *crystal* dataset. Again the values are the worst measured across all nodes, since the variance between the nodes is at most 1 ms. The latencies introduced by the MPI communication, the reordering of network messages and the assembly of the encoded frames are well below 1 ms and are therefore omitted from the figure. However, all operations and communication are included in the *Complete process (source)* column and the *Complete process (display)* column. While the reordering latency is usually extremely low it can increase when a packet is missing, which rarely happened during the tests, but is limited to the maximum duration described in subsection 4.2.7. The overall latency between a frame being captured and finally being displayed is the sum of *Complete process (source)*, the *Network* latency and *Complete process (display)*. For the other three datasets, *solvent*, *mdao* and *laser*, the average, minimum and median latency for the encoding is also around 6 ms, while the maximum latency is between 8 and 10 ms. Additionally, the encoding latency is roughly the same for all three settings, only the maximum latency increases for the high setting. The same is true for the colour conversion and rotation of the captured frame,



**Figure 4.11** – The measured throughput for the four visualisations that were looped multiple times. The *mdao* (top left) requires the least bandwidth, while the *crystal* (top right) requires the highest, with up to 2.5 Gb/s. The other two datasets, *laser* (bottom left) and *solvent* (bottom right), use at most 785 Mb/s and 600 Mb/s. The first peak for the *solvent* dataset can be considered an outlier as it does not occur again and is due to the fact that we sent 30 key frames after a client is detected in order to make sure that a key frame is received so the decoder can start. This behaviour can also be seen for the *mdao* dataset where a small peak is visible at the start of the measurement. It is clearly visible that the high encoder setting (■) requires significantly more bandwidth than the medium setting (■), which in turn requires significantly more bandwidth than the low setting (■).

for which the latency is between 0.5 ms and 4 ms with the median being 2 ms. However, the maximum latency for the conversion and rotation can increase if the GPU is under heavy load. For example, the *laser* dataset increases the maximum latency from 4 ms to around 30 ms therefore, reducing the number of captured frames. The major impact on the overall latency is usually the encoding and the mapping after the decoding, which allows accessing the decoded frame, which takes 5 ms on average but can take up to 19 ms while using the high setting. Note that the mapping is included in the Decode column in Figure 4.10, as it is technically part of the decoding process. The time the decoded frame is in the display queue is at most 1 ms and usually far below that if the rendering is not limited. The maximum latency increases if Quadro Sync, or the MPI barrier, is used to 1000/refresh rate ms, since the rendering is limited to the refresh rate of the display. Overall the average end-to-end latency is approximately 14 ms, while the maximum is 37 ms with the exception of the *laser* dataset where the maximum

**Figure 4.12** — Graph of the measured throughput in Megabits per second for the transmission of the slides. Each peak signifies a change in the slides, e.g. an animation on a slide or a new slide, which is sent to the clients. The times between the peaks show that identical frames are not encoded and transmitted as the measured throughput is basically zero.

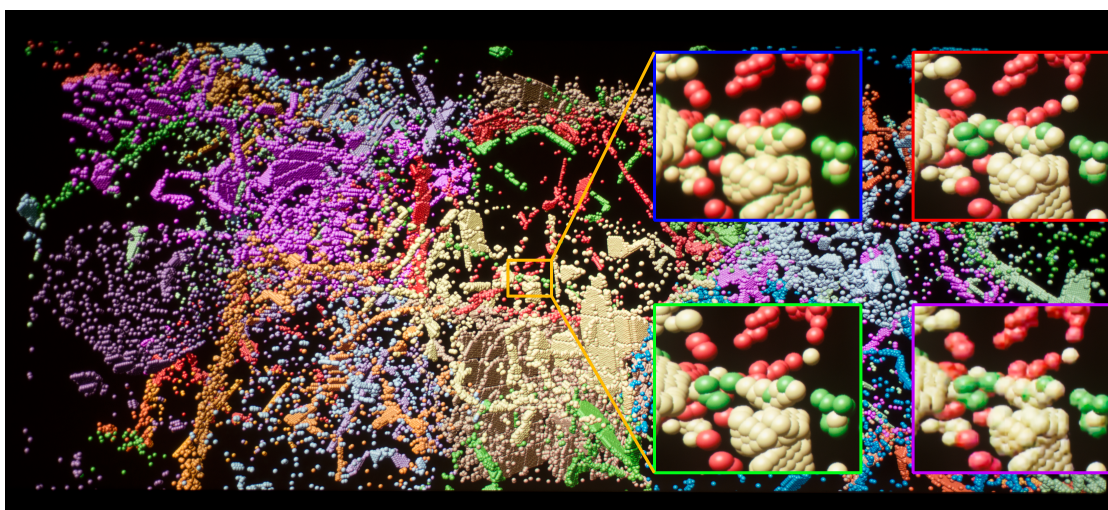


end-to-end latency is up to 50 ms, due to the longer maximum frame conversion latency. The minimum end-to-end latency is around 12 ms.

In addition to the latency the required bandwidth for this scenario was measured using each of the datasets in combination with the three encoder settings. Bandwidth was measured as the sum of all outgoing data per second converted to Megabits per second and is shown in Figure 4.11. As can be seen, the measured throughput differs between the four datasets, with the *crystal* dataset requiring 2.5 Gb/s for the high setting while the *mdao* only requires 290 Mbi/s using the same setting. Additionally the required bandwidth also differs between the settings with the low setting only using 60 Mb/s for the *crystal* and 8 Mb/s for the *mdao* dataset, while the medium setting uses 530 Mb/s and 30 Mb/s respectively. The measured throughput of the *laser* dataset is lower due to the fact that on average only 25 frames are captured per second, due to the increased frame conversion latency.

The required bandwidth for the video conference scenario can vary greatly due to encoder settings for the video camera, the resolution of the video camera, etc., but it never exceeded 200 Mb/s during the testing. A reasonable compromise for the encoder settings for the 4K camera in the video conference scenario required about 30 Mb/s. The transmission of the presentation slides makes use of equal frame detection and thus transmits only new slides to the clients. This reduces the required bandwidth significantly, as can be seen in Figure 4.12, even though encoder settings similar to the high setting in the screen sharing scenario were used. The maximum bandwidth measured for the slides was around 70 Mb/s while showing an embedded video. The display of the full HD slides on the ten display nodes at site A consumed around 7 Gb/s on the InfiniBand network. Since the network can handle this high bandwidth no encoding was used for the local presentation of the frames, which results in a median display latency of around 5 ms. Encoding slides with a resolution of  $1920 \times 1080$  pixels would have added additional latency. Audio samples could be encoded in, at most, 1 ms (0.2 ms on average) and decoded in less than 1 ms (0.09 ms on average). The end-to-end transmission of audio samples took a maximum time of 2 ms in the arguably very good network situation, way below the ITU-recommended maximum of 150 ms for voice





**Figure 4.13** — A photo of the entire *crystal* dataset on the display of site A and four close-ups of the same area of the display. There is no visible difference between the original rendering and the streamed version of the medium and high setting. The photo of the low settings shows that this setting produces an image that is of a lower quality than the others. The orange rectangle shows the part of the screen that can be seen in the close-ups. The blue border shows the rendering in native resolution, the red border shows the result using the medium encoder setting, the green border shows the result using the high encoder setting and the purple border shows the result using the low encoder setting.

communication. The required bandwidth for the audio stream was around 0.13 Mb/s.

The system has been specifically designed to move large numbers of pixels efficiently, which is a viable solution when exploiting the full potential of state-of-the-art hardware. The sharing of external screens, such as a laptop, is based on signal capturing cards, since such devices have become affordable, and because this approach leads to a unified software design in a system that is based on moving encoded pixels. Additionally, this solution offers the best user experience resembling a standard video projector.

Since the maximum required bandwidth for reasonable encoder settings is well below the limit of 1 Gb/s, the system can be used in a realistic setting. The screen sharing of  $10800 \times 4096$  pixels at 60 frames per second is possible even at a lower available bandwidth. The low encoder setting yields visible artefacts, which are noticeable while standing close to the large display (see Figure 4.13). The medium encoder setting produced an image almost identical to the original, such that the difference is hardly visible for the spheres, while still using a realistically available bandwidth. With the

high setting no difference is noticeable, but the required bandwidth exceeds what is typically available over the internet. For the molecular datasets, consisting mostly of spheres, there is no need to use anything higher than the medium setting, since the difference in quality does not justify the required bandwidth. For visualisations that contain very fine structures (cf. Figure 4.5) it could still be beneficial to use a setting between the medium and the high setting to keep the image quality high and artefact free. Using the H.265 codec decreases the required bandwidth but increases the encoding latency. Therefore, this is a trade-off that can feasibly be made depending on which is the more important, lower latency or lower bandwidth.

Latency for all screen sharing scenarios was not noticeable in side-to-side comparison, even for the high quality setting. Usually the major contribution to the latency, with up to 10 ms, on the display side comes from the encoder. Note that these maximum latency values are outliers as the average and median values are significantly lower. However, if the GPU is under heavy load the frame conversion can introduce latencies of up to 30 ms since the compute shader needs to wait before it is executed. This reduces the number of frames captured from 60 to approximately 25 and in turn also the measured throughput, as can be seen for the *laser* dataset. In order to reduce the latency, the rotation can be performed on the client side and if the encoder supports BGRA colour input no colour conversion is necessary, making this step unnecessary. If this is not possible the frame conversion and the encoding can be performed on either a different GPU or a different node within the cluster. Both the downscaling and the tiling reduce the latency for the encoding. While this was an expected result for the downscaling, since the resolution is lower, it was not expected for the tiling, since the resolution stays the same but is distributed among multiple tiles. Using between two and 32 tiles the latency is lower than without using tiling. The assumption is that using 32 or more tiles increases the overhead from the context switch of the encoder and the time it takes to download each tile to outweigh the reduced latency of the encoding of each smaller tile. For more than 32 tiles the latency increases significantly for each additional tile. The latency reduction requires one encoder session per tile. If there are fewer sessions than tiles, the latency and throughput increases since each session has to encode multiple tiles and always uses key frames.

During the development of the system some solutions were explored that did not turn out to be viable. Most notably, the original thought of using RTSP streaming for the video conference part turned out to not ever achieve latencies below even 500 ms, which is too high for two parties to talk naturally. The custom UDP-based network layer solves this problem, albeit at the cost of interoperability. Furthermore, the software-based video encoders provided by Windows Media Foundation are barely capable of handling full HD streams on nodes handling more than one role or multiple streams. This leads to the conclusion, that video conferencing streams and external sources with 4K resolution in real time only work due to a software design that systematically makes

use of modern graphics hardware and zero-copy operations whenever possible.

As in any major software system, there are solutions that work, but that one would do differently now. Most importantly, the use of Windows Media Foundation did not solve as many problems as originally anticipated. The reasons for this are manifold, ranging from the fact that the initially targeted operating system was Windows 7, which has no access to hardware-powered encoder transform nodes in Media Foundation, through the need to support existing ASIO hardware and to the fact that the frame grabbers only use formats that are supported out-of-the-box if running with a resolution less than 4K. All of this led to a system that combines several implementation islands which are more or less tied into the initial Media Foundation core of the software. For instance, the ASIO adapter is closely integrated as a Media Foundation source, whereas the video encoder is an independent low-level implementation that directly uses the hardware video encoder, which is a daunting endeavour by itself. Likewise, the Opus-based audio path is not at all integrated with Media Foundation, which negates the benefit of Media Foundation synchronising the timestamps of all media sources involved. In hindsight, it would have been reasonable to either “go full Media Foundation” and implement everything as reusable media source, sink or transform, or to extract all information using sample grabbers at the earliest possible convenience and base all processing on a own implementation. This could also have prevented the presence of different sources of parallelism ranging from Media Foundation queues, the thread pool of the IOCP over the live555 worker to threads dedicated to the Opus transcoder. The video conference setup uses a dedicated node for capture, audio playback and for transmitting and receiving audio and video streams. As incoming video streams are relayed to the display nodes via MPI, the initial expectation was to delay audio playback to be synchronous with the video stream. However, this turned out to be unnecessary for the current setup. Likewise, many low-level optimisations that were made are probably unnecessary for a working solution. As an example, initially, lock-free means of moving data between video-related threads via a triple buffer were used, to avoid the cost of system-level synchronisation primitives. This turned out to be impractical, because triple buffers do not enforce that all nodes in the cluster actually draw the same frame, even if the buffer swap is synchronised. Therefore, that implementation was replaced with a more conventional one using mutexes, which in the end had no noticeable impact on the performance of the system.

The Powerwalls for which the system was developed run on Microsoft Windows, making it a natural choice to leverage its coherent media processing capabilities. However, the building blocks for the software, like the video encoding and decoding SDKs and capturing SDKs, are also available on Linux, making the concept transferable to this system. For example the NVIDIA Capture SDK provides equivalent capabilities to the Desktop Duplication API for compatible hardware. Developing a system with similar capabilities therefore merely requires the effort of implementation.



## 4.5 Summary and Conclusion

The system discussed in this chapter offers different remote collaboration scenarios on large high-resolution displays based on pixel streaming using modern graphics hardware, which grew out of the need to support these scenarios in a collaborative research project in visual computing. The system supports collaboration across sites by sharing the content of one large display to others at its native resolution in real time. This is application-agnostic, i.e. it works for any visualisation that can be displayed on the respective tiled display without changing the code of the application. In addition to this scenario, the system supports real-time video conferences and multi-site talks that complement the video stream with a stream of the output of a laptop. Key to these types of applications is the multi-threaded and hardware-centric design of the software, which makes this solution viable even if one of the sites has only Gigabit Ethernet connectivity.



## ALGORITHMIC OPTIMISATIONS

### Parts of this chapter have been published in:

- F. Frieß, M. Landwehr, V. Bruder, S. Frey, T. Ertl. Adaptive Encoder Settings for Interactive Remote Visualisation on High-Resolution Displays. Proceedings of the IEEE Symposium on Large Data Analysis and Visualization - Short Papers, pages 87–91, 2018. [45]
- F. Frieß, M. Becher, G. Reina, and T. Ertl. Amortised Encoding for Large High-Resolution Displays. IEEE 11th Symposium on Large Data Analysis and Visualization, pages 53–62, 2021. [43]

The system discussed in chapter 4, as well as other systems for pixel streaming based remote visualisation, has to make compromises between quality and latency as well as throughput due to bandwidth limitations. They either uniformly decrease the quality to maintain adequate frame rates at a low throughput, or deliver a high quality image at lower frame rates and at a very high throughput and therefore induce bandwidth requirements that cannot generally be met. In this chapter, methods to reduce the required bandwidth while keeping the image quality high are discussed. The focus of these methods is on optimisations to the encoding that do not require any user input, such as changing the quantisation of certain macroblocks or down-sampling prior to the encoding. The presented methods make use of GPUs to accelerate the necessary computations in order to achieve interactive frame rates.

The bandwidth limitations of remote visualisation have been previously addressed. Both Bolin and Meyer [19] and Levoy [93] proposed adaptive sampling techniques,

while Koller et al. [82] and Herzog et al. [61] used image compression techniques. In order to efficiently compress and stream images of dynamic 3D models Pajak et al. [113] use augmented video information. The approach by Moreland et al. [108] uses level-of-detail techniques in order to provide an interactive rendering regardless of the network performance. Frey et al. [42] presented a technique that is targeted towards scientific visualisation in a remote setup. In order to optimise image quality they integrate sampling and compression techniques to balance visualisation and transfer.

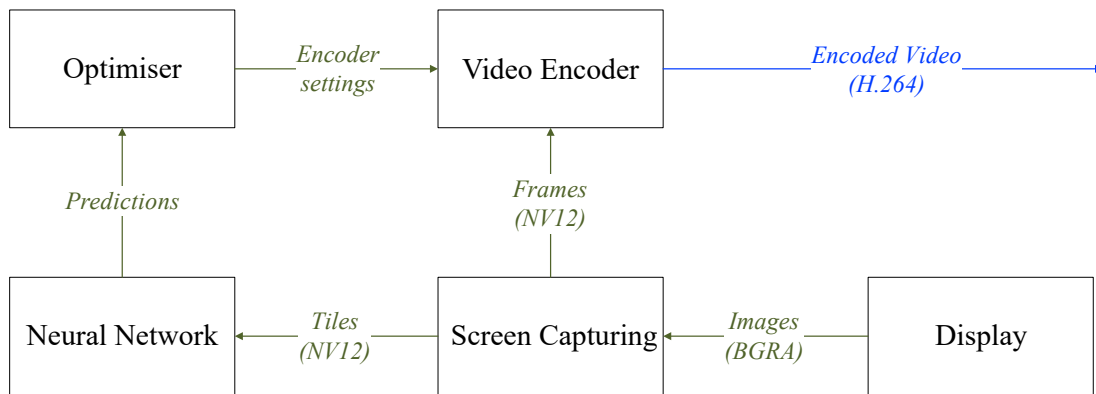
Section 5.1 describes a method that uses the predictions of a CNN to change the quantisation of each macroblock in order to reduce the required bandwidth for transmitting a full high-resolution display image. For this, the rendered frame of each display node is split into several equally large tiles. Then, the CNN is used to predict the size of each tile after encoding as well as the quality in terms of similarity to the original image. The size and quality is predicted for three different encoding settings. Based on the prediction, the encoding process is optimised for maximum quality, i.e. similarity to the original image, under the constraint of the available maximum bandwidth.

Section 5.2 describes a method that adapts the concept used by amortised rendering techniques for video encoding in order to reduce the encoding latency and the bandwidth needed to transmit a frame over the network. Instead of encoding at full resolution, every frame is sub-sampled, periodically alternating between four different patterns and then uses a temporal upscaling to reconstruct the original image, after decoding. Static parts are reconstructed by simply copying the colour value of the corresponding pattern, while non-static parts are reconstructed by only using the last received frame and interpolating the colour values of the missing pixels.

## 5.1 Adaptive Encoding

Convolutional neural networks have proven to be very good at classification and localisation tasks [85, 63], but can also be used to predict the quality of images. Several approaches have been proposed for this task. Li et al. [94] developed an image quality assessment algorithm that utilises a regression neural network. They use their technique to predict image quality that is relative to human subjectivity by applying a range of different distortion types. Kang et al. [76] use a CNN to predict image quality without a reference image, instead they use image patches as input to their network. Furthermore, they combine feature learning and regression as well as an optimisation process to estimate image quality in terms of human perception. A similar approach was proposed by Bosse et al. [20]. They use a deep neural network to predict image quality which works in a similar way to human perception, also feeding image patches into the network.

While all of these techniques use neural networks for image quality predictions, their



**Figure 5.1** — Overview of the part of the system (cf. Figure 4.4) that was changed for the adaptive encoding. Two additional parts, the Neural Network and the Optimiser were added to the pipeline. The captured frame is now converted to the NV12 colour format and forwarded to the video encoder. Additionally, the screen capturing outputs tiles, each containing only the luminance values of a part of the full frame. These tiles are used by the neural network to predict the quality and size after encoding for the three encoder settings (cf. subsection 4.2.5). The predictions are then forwarded to the optimiser that tries to find the best combination of encoder settings for the tile so that the resulting size is below a given threshold. The settings are then forwarded to the encoder and combined with the frame resulting in an encoded frame that uses different settings for each tile.

goal is different. The objective of the adaptive encoding is to use the predictions in order to determine the optimal encoder setting with respect to a bandwidth limit, while their main goal is to judge the perceptual visual quality. Based on the predictions, the encoding process is optimised for maximum quality, i.e. similarity to the original image, under the constraint of the available maximum bandwidth.

### 5.1.1 Method

The algorithm adapts the pipeline, see Figure 5.1, from capturing a frame to encoding it on the server side of the system described in chapter 4, by adding two new stages. The first stage is the CNN that receives the last captured frame, converted to grey values, separated into tiles and which predicts the image quality and size of the encoded frames for each of the three encoder settings described in subsection 4.2.5. The second stage is the optimiser that determines the QP value of each macroblock based on the predictions of the CNN and the maximum allowed bandwidth. All macroblocks that are inside the

same tile are assigned the same QP value.

The first step on the server side is the capture of the last rendered frame as a texture. This texture is then converted to the NV12 colour format using a compute shader that also, optionally, rotates the frame in 90° steps. It outputs a full resolution texture, which will be used by the encoder, and multiple tiles, each containing a part of the full resolution texture, which are used by the CNN. The CNN then predicts the quality and size for each tile and forwards this to the optimiser, which tries to find the optimal setting for each tile to remain below a given bandwidth threshold. Both, the full resolution frame and the result of the optimiser, are given to the encoder. After the encoding is complete the resulting encoded frame is sliced and transmitted to the client via UDP. The threshold of each node is then updated based on the total amount of leftover bandwidth, i.e. nodes that did not require the whole allocated bandwidth distribute the leftover bandwidth to nodes that required more bandwidth. These steps are processed, in parallel, on all nodes that render a part of the (potentially distributed) frame buffer.

The steps on the client side are again processed in parallel on all nodes that render a part of the (potentially distributed) frame buffer. In the first step the incoming slices are reordered and assembled before decoding. After the frame has been decoded it is queued for display.

The following paragraphs provide additional details of the differences between the pipeline of the adaptive encoding and the pipeline of the baseline system discussed in chapter 4.

**Screen Capturing** Compared to the baseline system the compute shader that converts the colour format of the last captured frame has changed significantly. It now only converts to the NV12 colour format and produces two outputs. The first is the full resolution texture, while the second is an array of texture tiles that each contain a part of the full resolution texture. The first texture is directly passed onto the encoder, while the tiles are forwarded to the CNN for the predictions. While the colour conversion and the output are different, the shader still uses the same optional rotation in 90° steps prior to the colour conversion as the baseline system.

**Neural Network and Optimizer** The CNN uses the tiles in order to predict the quality and size of each tile when it is encoded using any of the three settings described in subsection 4.2.5. These predictions are then forwarded to the optimiser that uses a greedy algorithm to determine the optimal setting for each tile in order to stay below a given bandwidth threshold. This results in different settings used for each tile. Based on this setting the QP value of all macroblocks inside the tile is forced to the QP value of the setting. For example, a tile encoded using the medium setting forces all macroblocks inside the tile to use a QP value of 31.

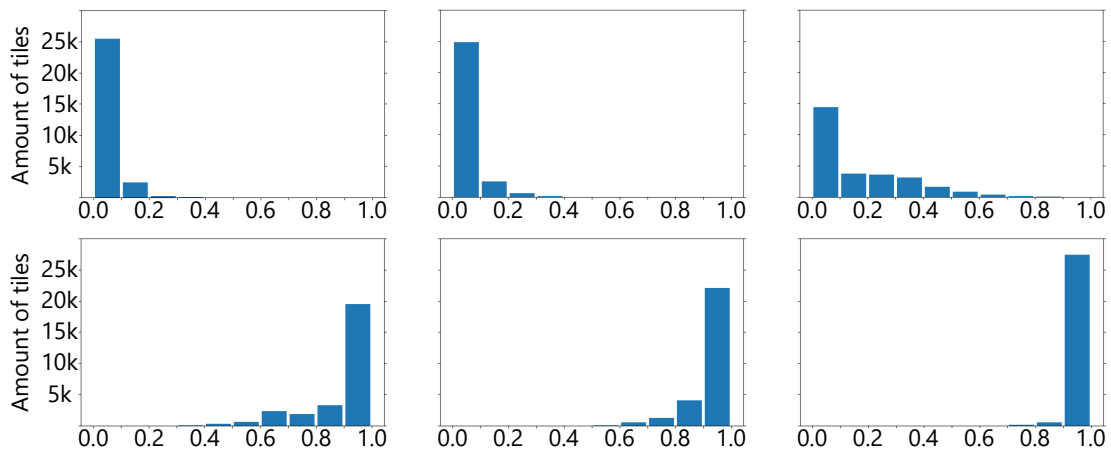
**Encoding** The encoding has only changed slightly, it now uses an offset map containing a value for each macroblock of the full resolution frame that changes the QP value from 51 to the value determined by the optimiser. Apart from this exception, the encoding uses the constant quantisation mode, which sets the QP value of every macroblock to 51 prior to the offset map, does not limit the bitrate and uses an infinite GOP length with a regular intra refresh as error resilience measure, similar to the baseline system.

### 5.1.2 Implementation Details

The adaptive encoding uses the implementation of the baseline system described in chapter 4 and changes only small parts of the software. These changes include the new Optimizer as well as the CNN and changes to the compute shader that used during the screen capturing step. The tiling, performed by the compute shader, introduces a GPU-to-GPU copy, and the QP offset map is copied from the CPU to the GPU after the Optimizer is finished.

**Screen Capturing** The original system uses the Desktop Duplication API [105] to gain fast access to the full desktop as a texture in the BGRA colour format. This texture is then converted, to whichever colour format the encoder expects, and optionally rotated, downscaled or tiled to fit the maximum resolution of the encoder, using a compute shader. For the adaptive encoding a new shader was implemented that only converts from BGRA to NV12 and generates two outputs. The first is the, optionally rotated, full resolution texture using the NV12 colour format and the second is an array of smaller textures, only containing grey values, i.e. the luminance part of the NV12 colour format. Each of the smaller textures, also optionally rotated, is a tile of the full frame with a resolution of  $240 \times 512$ . The conversion to NV12 is carried out since the CNN works with grey value input, which is equivalent to the luminance part of the NV12 colour format. This provides the added benefit that the encoder does not have to convert the colour itself and can use the already converted full resolution texture directly. The texture is then passed on to the encoder, while the texture array is forwarded to the CNN.

**Neural Network and Optimizer** The major part of the algorithm is to find an optimal encoding setting for each tile based on the quality and size. In order to achieve this, each tile needs to be analysed and checked to find out if it contains a lot of fine structures. For that purpose, an algorithm that is able to predict the quality of images after encoding as well as the size of the encoded tile is required. Therefore, a CNN-based regression to predict the size and quality of a tile for each of the three encoder settings is used. The output of the network is a vector containing six values, one for the expected



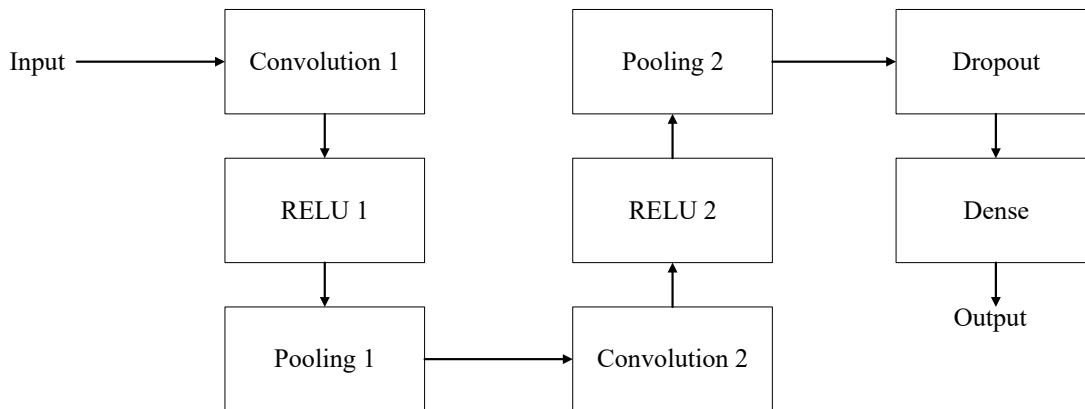
**Figure 5.2** – The distribution of the normalised tile sizes (top row) and the SSIM values (bottom row) for the encoding settings for all tiles used during the training of the CNN. The rows show the low, medium and high setting from left to right. As can be seen, the normalised size increases for the medium and high setting compared to the low setting. Additionally, the SSIM value distribution decreases since more tiles are closer to 1, i.e. close to the original tile. © IEEE 2018 [45].

quality and one for the expected size after the encoding with each of the three settings. The encoder setting used for each tile is decided based on the predictions and the greedy optimiser.

To assess the image quality the structural similarity index (SSIM) [166] is used by comparing the original to the encoded image. SSIM takes the luminance, contrast and also the structures of two images into account in order to compute a numerical closeness indicator. This indicator can be rescaled into the range  $[0, 1]$ , with 1 representing an identical image and 0 the complete opposite.

To train the network 22560 training images from different volume rendering simulations and 5632 evaluation images were used. Each of these images was divided into tiles with the size of  $240 \times 512$  and then each tile was encoded three times using each of the three different encoder settings. For each of those three, the SSIM values with respect to the original tile were computed as well as the size of the encoded tile. Both were used as labels during the training phase of the network. Figure 5.2 shows the SSIM value distribution of the training data. It is clearly visible that, for the high encoding setting, the SSIM values are between 0.95 and 1.0, while most tiles are between 0.99 and 1.0. For the other two settings the range is greater and fewer tiles are close to 1.0, i.e. equal to the original tile.





**Figure 5.3** — Overview of all layers of the convolution neural network used to predict the quality and size of the tiles. The input of the network are the grey value tiles in batches of  $t * 240 \times 512$  pixel, with  $t$  being the number of tiles in the batch. The overall network is deliberately kept simple in order to reduce the time needed for predictions. © IEEE 2018 [45].

The network consists of two pairs of convolution and pooling layers as shown in Figure 5.3, followed by the dropout and dense layer. The network is deliberately kept simple in order to reduce the time needed to predict the encoder setting for all tiles. The input of the network are the tiles, with a resolution of  $240 \times 512$ , and the output is a vector with six values containing the predicted SSIM and size values for each encoder setting. To further reduce the time needed for predictions, the tiles can be combined in batches, i.e. combining multiple tiles into one bigger input. This batch is of the size  $t * 240 \times 512$  with  $t$  being the number of tiles in the batch. The first layer (Convolution 1) performs a convolution with a kernel size of  $7 \times 7$  and uses 32 kernels, then a RELU (RELU 1) activation function is applied, resulting in a feature map of size  $32 \times 240 \times 512$ . The first pooling layer (Pooling 1) performs a maxpooling with a kernel of size  $4 \times 4$  pixel and stride of 4, effectively reducing the size to a batch of  $32 \times 60 \times 128$ . This allows the network to still detect features that are in the general area of an image patch instead of requiring it to be at the exact location, therefore speeding up the computation time by reducing the size of the feature map. Afterwards, the second convolution layer (Convolution 2), as well as a RELU (RELU 2) activation function, is applied. This time with 64 kernels with the same size as in the first convolution layer, resulting in a feature map of the size  $60 \times 128$ . The second pooling layer (Pooling 2) applies a  $4 \times 4$  kernel reducing the size to  $64 \times 15 \times 32$ . In the first dense layer the output of the second pooling layer is flattened to a one dimensional vector containing 2048 units of 1 pixel each. Additionally, a dropout layer drops 20% of the activations to prevent overfitting. The

last layer is another dense layer that condenses the vector to the desired output size of six values.

**Optimisation of Encoder Settings** The goal of this step is to optimise the quality of the tiles for a given size threshold  $T$ , i.e to have the highest possible encoding setting per tile while keeping the overall size of the tiles below the threshold. This is equivalent to the multiple-choice knapsack problem and can be optimally solved by minimising the objective function:  $minimise \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij} * (1 - SSIM_{ij})^2$ . With  $N$  being the number of tiles and  $M$  the number of encoding settings. The first constraint arises from the fact that exactly one setting for each image tile can be taken:  $\forall_j \in N : \sum_{i=0}^{M-1} x_{ij} = 1, x_{ij} \in \{0, 1\}$ . The second constraint restricts the overall size of all tiles to be less or equal to the given threshold:  $\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij} * SIZE_{ij} \leq T$ . Since this problem needs to be solved faster than the integer linear program, a greedy approach that approximates the optimal solution was chosen. For this, first the quality is optimised locally on each display node by taking all the predicted SSIM and size values into account. The optimiser first sets the encoding setting of all tiles to high and sorts the tiles, in descending order, according to the SSIM value of the medium encoding setting. Then for all tiles that have an SSIM value bigger than the defined threshold of 0.975 the encoding setting is reduced to medium. After that, the tiles are sorted again in descending order, according to the SSIM value of either low or medium, depending on the current encoding setting of the tile. This process is repeated until there are no tiles left where the encoding setting can be reduced without losing too much quality. Then, the overall size of the tiles is computed based on the predicted sizes and the assigned encoding settings. If this value is above the size threshold, the encoding settings of the tiles are reduced until the condition is met. For this, the optimiser uses the tiles with the lowest difference in the predicted SSIM values between the current encoding setting and the setting one level below that. This ensures that the best possible quality is kept while staying below the given size threshold. The last step is to compute the difference between the threshold and the required number of bytes and to distribute this difference across all nodes via MPI. Nodes that did not require all of the allocated bandwidth distribute their remaining bandwidth to nodes that required more.

**Encoding** For the adaptive encoding an offset map that changes the QP value of each macroblock from 51 to the value determined by the optimiser is required. This map is used by the encoder to change the QP value of a macroblock from that constant value, which is set to 51, to the desired value and needs to be provided together with a corresponding frame. During a pre-computation step a map is created that links the position of all macroblocks in the QP offset map to the ID of the tile that contains the macroblocks. Then, after the optimiser step, the QP offset map can be computed by iterating over the tiles and assigning the determined QP value to the macroblock. Since

**Table 5.1** — Analysis of the accuracy, denoting the mean squared error (MSE), the average absolute error (AAE), the standard deviation (STD) of the AAE and the maximum AAE, of the trained CNN for predicting quality (SSIM) and tile size.

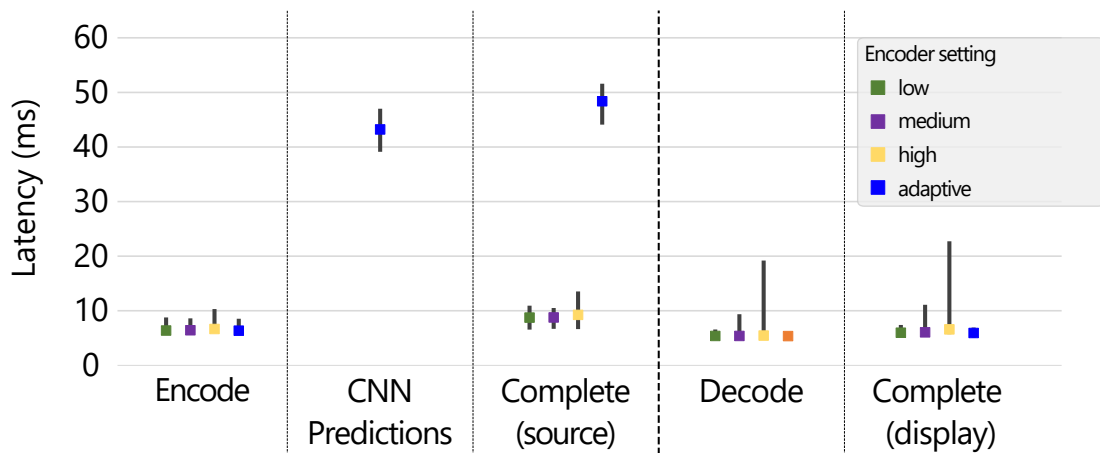
<b>SSIM</b>	MSE	AAE	STD	max AAE
low	$2.43 * 10^{-5}$	0.0026	0.0041	0.0576
medium	$1.19 * 10^{-5}$	0.0018	0.0029	0.0308
high	$5.38 * 10^{-6}$	0.0016	0.0016	0.0156
<b>tile size</b>	MSE	AAE	STD	max AAE
low	$7.95 * 10^{-5}$	0.0046	0.0076	0.1134
medium	0.0001	0.0049	0.0093	0.1566
high	0.0011	0.0180	0.0281	0.2279

neither the AMD nor the Intel SDK provide a function to alter the QP values of the macroblocks using such a map, only the NVIDIA SDK and the CPU encoder SDKs can be used. While it would be possible to implement the same functionality using a dedicated encoder for each setting and passing the tiles to the corresponding encoder, it would require each tile to be encoded as a key frame. Another option would be to create one encoder session for each tile and reconfigure the encoder based on the setting determined by the optimiser. Therefore, the usage is restricted and it requires an SDK that provides this kind of offset map, since the alternative options would increase the latency, as well as the bandwidth compared to the offset map solution. Similar to the baseline system the encoder session uses its own ring buffer, containing the Direct3D 2D interop textures that are used by the compute shader to output the full resolution frame, and works either asynchronously or synchronously, depending on the SDK.

### 5.1.3 Results and Discussion

A quantitative evaluation of the approach was performed by measuring the latency and throughput required to share the content of a tiled display in a local area network setting, as described in section 4.3. The throughput and latency was measured using four different visualisations, also described in section 4.3, and then compared to the system described in chapter 4. The focus of the evaluation was on the following questions: Does the adaptive encoding have a negative effect on the encoding and decoding latency? By how much can the required bandwidth be reduced when using adaptive encoding?

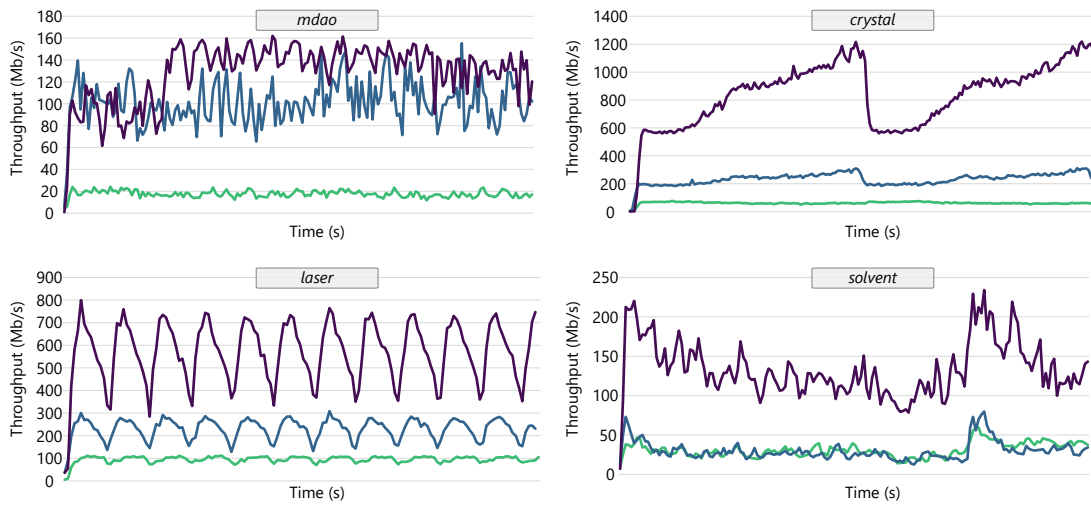
**Test Scenarios** In order to make the tests comparable to the baseline system the same settings were used: the low setting uses a quantisation value of 51, the medium setting a value of 31 and the high setting a value of 11. Both, the normal and the adaptive encoding, were tested using each of the four visualisations and each visualisation was



**Figure 5.4** — The measured latencies, in milliseconds, for the steps of the encoding and decoding pipeline that differ the most between the adaptive encoding and the normal encoding. Similar to Figure 4.10 the plot shows the median values (coloured block) as well as the minimum and maximum latencies, indicated by the grey lines. There is a no difference for the encoding latency but the maximum decoding latency is reduced significantly. However, the CNN prediction introduces latencies of around 40 ms, increasing the overall latency (*Complete (Source)*) on the encoder side to around 50 ms.

tested once for each encoder setting. The adaptive encoding was tested three times per visualisation using a maximum bitrate of 10 Mb/s, 100 Mb/s and 350 Mb/s for each display node, for an overall bitrate of 100 Mb/s, 1.0 Gb/s, 3.5 Gb/s.

**Test Results** Table 5.1 shows the mean squared error (MSE), the average absolute error (AAE), the standard deviation (STD) of the AAE and the maximum AAE for both, the SSIM values and the tile sizes. It can be seen that MSE and AAE decrease for the SSIM predictions with higher quality, but increase for the tile size predictions with higher quality. The main reason for this is the range of occurring SSIM values, which are close to 1.0. The tile sizes on the other hand are mostly located at the lower end of the spectrum, due to the normalisation process, and have a bigger pool of potential values which also results in higher error values (see Figure 5.2). For all six values the approximated error rate is roughly around 5%, therefore, the network is by no means perfect, but the results are sufficient to calculate the possible SSIM values and tile sizes given the fact that the predictions need to be made as quickly as possible. The approximated error, which takes the potential value range and the average absolute error into account, was calculated since it is not possible to compute the accuracy for a



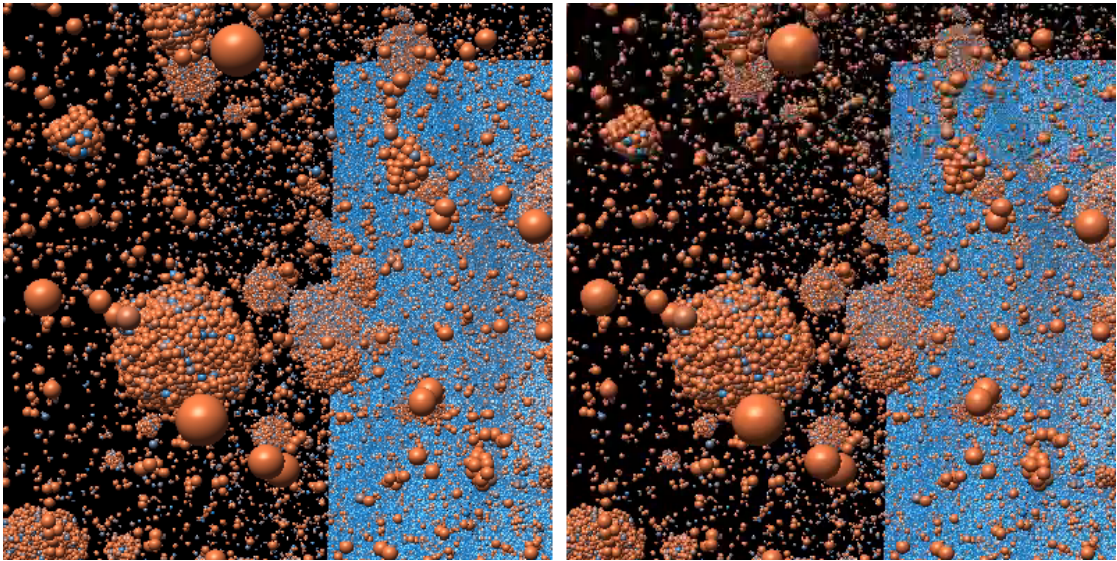
**Figure 5.5** — The measured throughput for the four visualisations that were looped multiple times. The normal encoding, which was limited to 25 frames per second, as well as the adaptive encoding using the maximum bandwidth of 350 Mb/s per node, result in a similar throughput, denoted in ■, therefore only the measured throughput of the normal encoding is shown. The adaptive encoding using 100 Mb/s (■) and 10 Mb/s (■) reduced the required bandwidth significantly for all datasets. However, for the *mdao* dataset (top left) there is little difference between the 350 Mb/s and 100 Mb/s maximum due to the fact that even for the 100 Mb/s maximum most of the nodes can still predominately use the high encoder setting for most of the tiles.

regression model directly.

The latency for the encoding, decoding and other major steps of the pipeline is unchanged when compared to the baseline system, see Figure 5.4. However, the neural network prediction adds around 40 ms of latency to the server side and the optimizer adds another 0.01 ms. Therefore, the *Complete (Source)* latency increases from about 10 ms to 50 ms. The median decode latency does not change and is about 6 ms while the maximum decode latency drops to 6 ms from 19 ms for the high setting due to the fact that not all macroblocks are encoded using the high QP value. This reduces the captured frame-rate from 60fps to about between 20 and 25fps.

Due to the difference in the captured frames the tests for the normal encoding were repeated using 25fps, which reduces the overall bandwidth by a factor of approximately two. Using the high setting for the normal encoding produces a similar result to the adaptive encoding using a maximum of 350 Mb/s per node since all tiles use the high encoder setting. Therefore Figure 5.5 only contains the results for the adaptive encoding

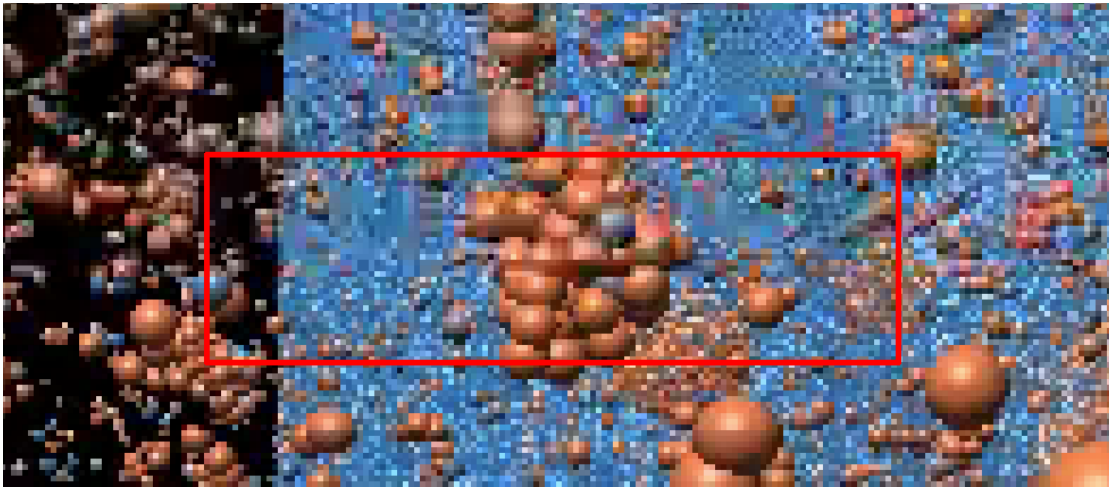




**Figure 5.6** — Comparison between the high encoder setting (left) and the adaptive encoding for a cropped part of the whole display. Differences between the two images only become apparent at high zoom levels (cf. Figure 5.7). © IEEE 2018 [45].

using 10 and 100 Mb/s per node. As can be seen, the measured throughput did not exceed the maximum bandwidth which clearly shows that the predictions are accurate. For the *crystal* dataset the overall throughput for the 10 Mb/s maximum is below 80 Mb/s but frames still contain tiles using the medium and high setting. For example, one of the display nodes in the centre of the display encodes 67.5% of the tiles using the low setting, 20% of the tiles using the medium setting and 12.5% of the tiles using the high setting. Using the 100 Mb/s caps the overall throughput at 300 Mb/s, well below the maximum of 1 Gb/s, while the same node encodes 35% of the tiles using the low setting, 52.2% of the tiles using the medium setting and 12.5% using the high setting. The distribution of the bandwidth amongst the nodes can be observed for the *mdao* dataset. Since the leftmost and rightmost node of the display require less than 1 Mb/s, due to the fact that each frame is completely black, they distribute the bandwidth to the node in the centre of the display. Therefore, the measured bandwidth for the maximum bandwidth of 350 Mb/s and 100 Mb/s is nearly identical. Overall, the adaptive encoding requires less bandwidth than the normal encoding using the medium and high settings, while it is on a similar level when compared to the normal encoding using the low setting.

To compare the resulting frame of the adaptive encoding against the normal encoding a single frame of the *laser* dataset was captured using the low, medium and high encoder

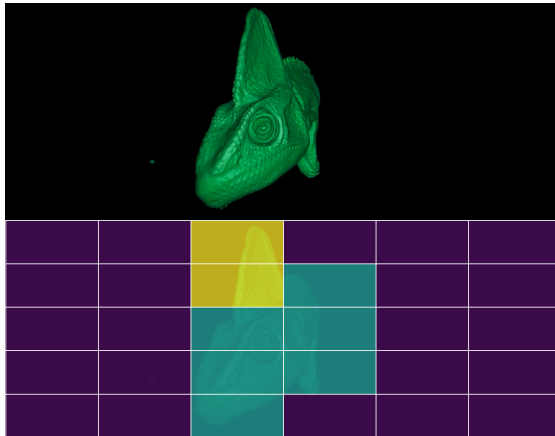


**Figure 5.7** — The boundary (highlighted by the red rectangle) between two tiles only becomes visible at a zoom level above 500%. The upper part of the frame contains a tile that was encoded using the low encoder setting, while the lower part of the frame contains a tile that was encoded using the high encoder setting. This leads to a high difference in quality for neighbouring pixels, producing the visible hard cut between the two tiles. However, this cut is not clearly visible without zooming. © IEEE 2018 [45].

settings as well as the adaptive encoding. For all of these frames the SSIM was computed with respect to the original rendered frame. As expected, the low encoding setting does not preserve many details and the final image has an SSIM value of 0.90. The medium and high setting perform much better when it comes to preserving the details of the visualisation with an SSIM value of 0.92 and 0.96 respectively. The adaptive encoding preserves most of the details and the resulting SSIM value is 0.93. Figure 5.6 shows a comparison between the high encoder setting (left) and the adaptive encoding. For that part of the frame there is no visible difference between the two because the adaptive encoding exclusively chooses the the medium or high encoder settings for the tiles in that part of the image. Furthermore, the boundary between two tiles using the medium and high settings is barely visible, while the boundary between low and medium or low and high settings is only visible at very high zoom levels, see Figure 5.7.

The adaptive encoding produces an image that is of similar quality to the normal encoding using the medium setting. However, the overall latency on the server side is increased to approximately 50 ms, which is mostly due to the time it takes the CNN to predict the SSIM and tile size. On the client side the latencies are similar, except for the decoding where the maximum latency fell from 19 ms to approximately 6 ms. This is due to the fact that not all tiles are encoded using the high encoder setting. The





**Figure 5.8** — The encoding settings determined by the CNN and the optimizer. The image on the top is the last captured frame and the image on the bottom shows the tiles highlighted by the encoder setting that was used to encode each tile. Tiles highlighted in (■) are encoded using the low setting, as they only contain the black background. Tiles highlighted in (■) are encoded using the medium setting and tiles highlighted in (■) use the high encoder settings. © IEEE 2018 [45].

high prediction latency reduces the number of captured frames per second to about 25. Although this is slower than the 60 frames captured by the baseline system it is still interactive. One way to circumvent the high prediction latency would be to update the encoder settings for the tile only every third frame instead of every frame. The prediction could run parallel to the encoding, since the encoding is separate from the compute workload of the prediction.

The measured throughput was reduced for all four visualisations compared to the baseline system. In order to provide a fair comparison the baseline system captured 25fps as well. This reduced the measured throughput by a factor of approximately two. A similar throughput to the normal encoding using the high setting was measured when the maximum allowed bandwidth per node was 350 Mb/s for the adaptive encoding, since all tiles are then encoded using the high setting. When the bandwidth was limited to 100 Mb/s the measured throughput was lower than the throughput of the normal encoding using the medium setting, while the image quality was similar. Limiting the maximum bandwidth to 10 Mb/s reduces the throughput to the level of the normal encoding using the low setting. However, the image quality was better as some tiles still were still encoded using either the medium or the high setting. This can be seen in Figure 5.8, the tiles, highlighted in red, that only contain the background are encoded using the low setting. The tiles, highlighted in blue, that contain some parts of the volume are encoded using the medium setting and the remaining tiles, highlighted in green, are encoded using the high setting.

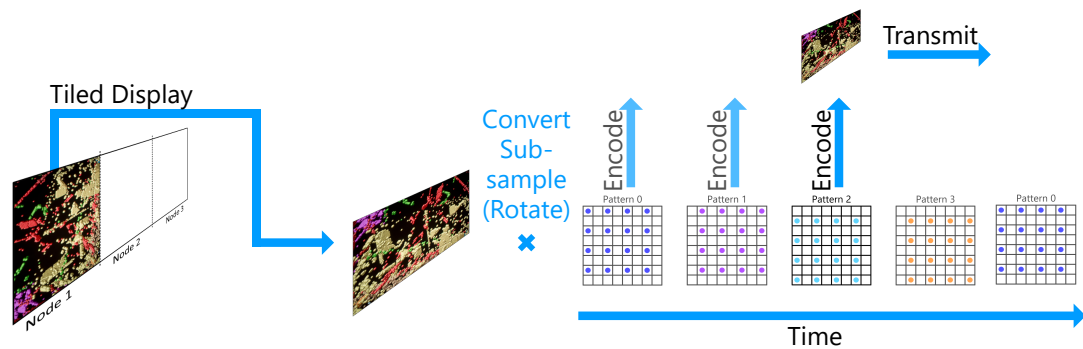
The boundaries between tiles of different settings produce artefacts but they only become clearly visible at high zoom levels, for example, 500% (cf. Figure 5.7). While this is not desirable, it is hardly visible while interacting with the visualisation. There is also a solution for the problem, namely introducing a gradient at the border of neighbouring

tiles. The gradient would increase/decrease the QP value at the border of each tile in order to remove the hard cut that is visible in Figure 5.7.

## 5.2 Amortised Encoding

Reducing the resolution of the input frames before encoding and then reconstructing the full resolution after decoding has been explored previously. Artem et al. [48] use convolutional neural networks for downscaling and upscaling, before encoding and after decoding respectively. The lower resolution reduces the required bandwidth for the transmission but also reduces the overall image quality, even in static regions of the image. Mariana et al. [3] proposed a framework that dynamically resamples the input frames spatially and temporally. During the encoding their framework uses a separate module to predict suitability of the spatial and temporal adaptation. Therefore, the encoder uses the resolution-optimized frames. After the decoding the frames are upsampled, spatially and temporally, based on the resampling decisions from the encoder. This upsampling is performed by a convolutional neural network super-resolution model. While their framework reduced the required bandwidth, it is limited to the High Efficiency Video Coding (HEVC) standard and adds a significant processing overhead. Keita et al. [155] analysed the super-resolution decoding from a theoretical perspective and showed that it performs better in low bitrate scenarios. Their analysis looks at non-temporal downscaling and upscaling and shows that the image quality is comparable to the full resolution encoding for lower bitrates, but worse for higher bitrates. The approach discussed in this chapter uses spatial subsampling with temporal coherence and is therefore able to offer a better perceived image quality as static regions are reconstructed perfectly.

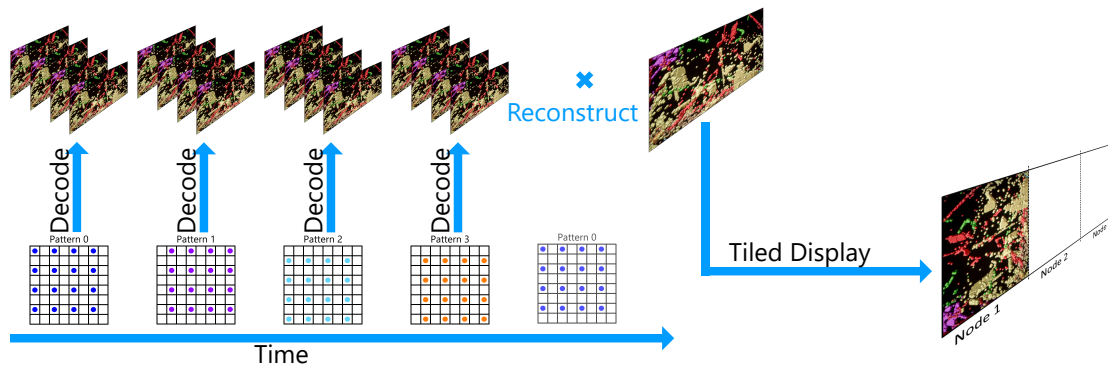
Reusing image information from previously rendered images has been explored for both offline rendering and real-time rendering, usually with the goal of either reducing rendering time or using super-sampling to increase image quality [2, 174]. The key idea is to make use of the spatio-temporal coherence present in most animation sequences and real-time renderings in order to amortise the rendering costs of a displayed image across several rendered frames. Amortised rendering techniques are frequently used in video games that often need to reduce internal rendering resolution in order to meet performance targets and then have to temporally reconstruct higher resolution output images using several internal frames [107, 38]. These concepts were adopted to amortise the time needed to encode a high-resolution display image and the bandwidth needed to transmit it across several transmitted frames of lower resolution. One of the key challenges of amortised rendering is the temporal reprojection of previously rendered frames to compensate for motion in dynamic scenes, which is possible for synthetic rendering as the complete scene information, such as camera position and world space coordinates of rendered pixels, is available. Video streams, however, usually do not



**Figure 5.9** — Overview of the major parts of the pipeline from capturing a frame on the server side to sending it to the client side. Each captured frame is converted, and optionally rotated, as well as sub-sampled by using the pattern associated with the current time. We cycle through the patterns, so every fourth frame uses the same pattern. The sub-sampled frame is then encoded and transmitted to the client. © IEEE 2021 [43].

contain this information, requiring the usage of a different solution for dynamic content. As an alternative to such temporal amortisation approaches, machine-learning-based upscaling of lower internal rendering resolutions has also been popularized in real-time rendering during recent years [171, 97]. However, many of these techniques also make use of additional output buffers, such as per-pixel depth and motion vectors, and are therefore not directly applicable to the video streaming scenario.

The adaptive encoding, discussed in the previous section, focuses on keeping the overall bandwidth below a given threshold. However, this increases the latency on the encoder side, due to the time it takes the CNN to predict the quality settings for each tile. The method described in this section adapts the concept, i.e. temporally reconstruct higher resolution output images using several internal frames, used by amortised rendering techniques for video encoding in order to reduce the encoding latency and the bandwidth needed to transmit a frame over the network. Instead of encoding at full resolution, every frame is sub-sampled, periodically alternating between four different patterns (cf. Figure 5.11) and then uses a temporal upscaling to reconstruct the original image, after decoding. This gives a perfect reconstruction of the static parts of the image, while moving parts are reconstructed only using the last received downscaled frame and therefore have a lower quality. The lower quality is barely noticeable given the targeted high resolution and the characteristics of the human visual system.



**Figure 5.10** — Overview of the major parts of the pipeline from decoding a frame on the client side to displaying the reconstructed frame. The displayed frame is reconstructed using the last four received and decoded downscaled frames for each pattern. We use two different reconstruction strategies, one for static and one for non-static parts of the image. For the static parts the newest frame for each pattern is used and the samples are copied into the corresponding display pixel. For the non-static only the newest frame is used and the samples are copied into the corresponding display pixel. Then, a bilinear interpolation is used to fill in the remaining display pixel. In order to detect the static and non-static regions we use the history, i.e. the last four frames of each pattern and compute the change over that time. © IEEE 2021 [43].

### 5.2.1 Method

The algorithm adapts the pipeline from capturing a frame to encoding it on the server side of the system, described in section 4.2, by changing the encoding step, see Figure 5.9. Instead of encoding the frame at full resolution, the amortised encoding cycles through four sub-sampling patterns to reduce the resolution to a quarter of the full resolution (cf. Figure 5.11). Each downscaled frame is then encoded using the H.264 video codec. The pipeline on the client side decodes the incoming downscaled frames and reconstructs the full resolution image, which is subsequently displayed, see Figure 5.10. For the reconstruction the colour change of a sample, i.e. the pixel of the downscaled frame, over the last four frames of every pattern is computed in order to detect static and non-static regions. Inside static regions the samples from all patterns are used, recovering the full image resolution, while inside non-static regions only the latest decoded frame is used for the reconstruction.

The first step on the server side captures the last rendered frame in a texture. As the encoder might expect a different colour format we use a compute shader to perform the conversion and optionally also perform a rotation in 90° steps. The rotation is necessary if the captured display is not in landscape mode and can be applied either

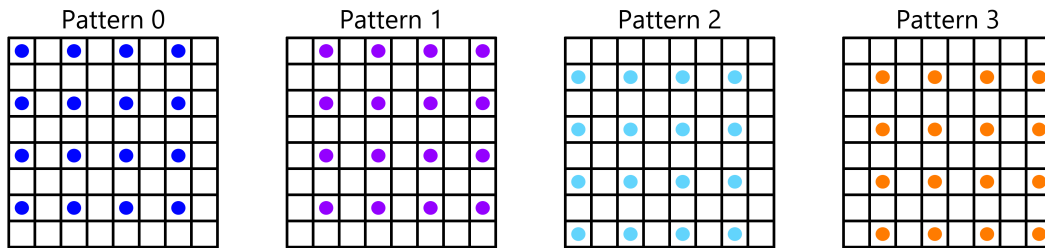
before encoding or during the rendering. This shader also applies the sub-sampling patterns and outputs a converted, optionally rotated, and downsampled texture. The downsampled texture is then forwarded to the corresponding encoder session, with one session for each pattern. After the encoding the resulting encoded frame is sliced and forwarded to the client via UDP. This step is processed in parallel on all nodes that render a part of the (potentially distributed) frame buffer.

The steps on the client side are, again, performed in parallel on all nodes that render a part of the (potentially distributed) frame buffer. Firstly the frames are re-assembled, from the received slices, and then queued for decoding. Again there are four parallel decoder sessions, one dedicated decoder session for each pattern. After the frames are decoded they are used to reconstruct the full resolution frame that is displayed once it is complete.

The following paragraphs provide additional details where the amortised encoding differs from the pipeline of the system discussed in section 4.1.

**Encoding and Decoding** The encoding and decoding is not changed compared to the baseline system, i.e both make use of the dedicated hardware en-/decoders that are separated from other workloads such as compute or graphics. The encoding uses the constant quantisation mode, does not limit the maximum bitrate and uses an infinite GOP length with a regular intra refresh as an error resilience measure. But instead of using one en-/decoder session per display node, the amortised encoding uses four sessions, one for each pattern. The decoded frames are copied into the display buffer, which holds a maximum of four frames for each pattern, instead of the display queue. These frames are used for the reconstruction of the full resolution frame that will be displayed.

**Reconstruction and Display** Simply reconstructing the full resolution frame by combining the last decoded frame for each pattern may cause the sub-sampling pattern to become visible, since four lower resolution frames captured at different times are used. While this is not a problem for static regions, which are reconstructed perfectly, it is a challenge for regions that contain motion. Therefore, all decoded frames are stored into the corresponding ring buffer, one for every pattern, with a length of four frames each. Based on this history, the colour difference over time is computed in order to detect static and non-static samples. If the colour value has changed more than a given threshold, the sample is considered to be non-static. In the static case every sample of the latest decoded frame for every pattern is copied directly into the corresponding display pixel. In the non-static case only the latest decoded frame is considered and again every sample is copied into the corresponding display pixel. The remaining display pixels, which would usually be filled from the other three patterns, are then interpolated based on the copied samples. After the frame is reconstructed, it



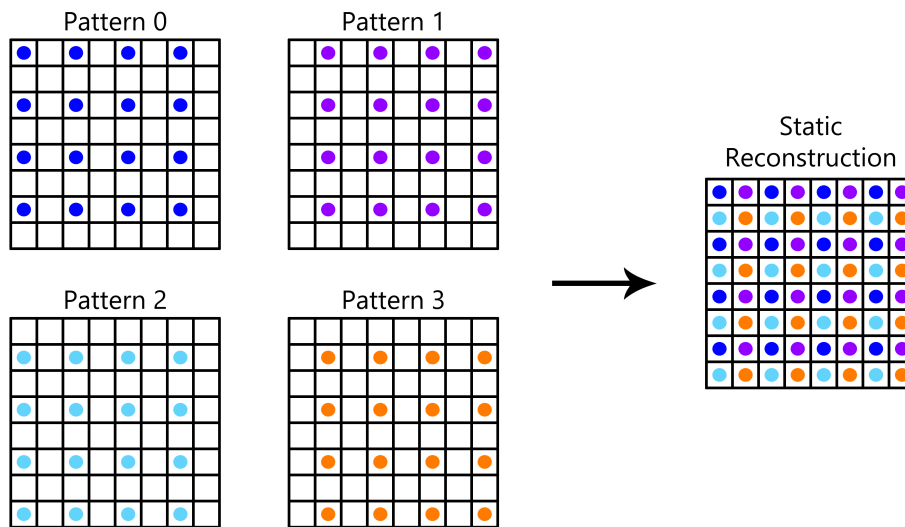
**Figure 5.11** — The four patterns ( $P_0, P_1, P_2, P_3$ ) used to downscale the full frame. They are applied successively, the frames at times  $t, t + 4, t + 8, \dots$  are sampled with the first pattern, the frames at times  $t + 1, t + 5, t + 9, \dots$  are sampled with the second pattern, the frames at times  $t + 2, t + 6, t + 10, \dots$  are sampled with the third pattern and the frames at times  $t + 3, t + 7, t + 11, \dots$  are sampled with the fourth pattern. © IEEE 2021 [43].

is rendered directly and therefore all scaling and positioning transformations, as well as colour transformations, are performed on the fly on the GPU.

### 5.2.2 Implementation Details

The amortised encoding changes the computed shader, used during the screen capturing step, and introduces a reconstruction step after the decoding. This introduces a GPU-to-GPU copy after the decoding since each frame is required for the reconstruction. In the following paragraphs, the integration and interplay of the amortised encoding in the full system is described.

**Screen Capturing** The original system uses the Desktop Duplication API [105] to gain fast access to the full desktop as a texture in the BGRA colour format. This texture is then converted, to whichever colour format the encoder expects, and optionally rotated, downsampled or tiled to fit the maximum resolution of the encoder, using a compute shader. For the amortised encoding a new compute shader was added that still performs the colour format conversion and the rotation. The original system used bilinear downscaling, or tiling to ensure that the resolution of the texture was below the maximum resolution of the encoder. The new shader performs the downscaling by applying one of the four sample patterns (Figure 5.11) and does not use the downscaling methods of the original. For each pixel  $(x_p, y_p)$  in the downsampled pattern frame, the corresponding pixel  $(x, y) = (x_p, y_p) * 2 + (idx \bmod 2, idx / 2)$  is selected in the captured frame, with  $idx$  being the index of the pattern, i.e 0, 1, 2 or 3. While capturing frames the shader cycles through the patterns by using the first pattern ( $P_0$ ) for the first captured frame, the second pattern ( $P_1$ ) for the second frame, and so on. This results in a new



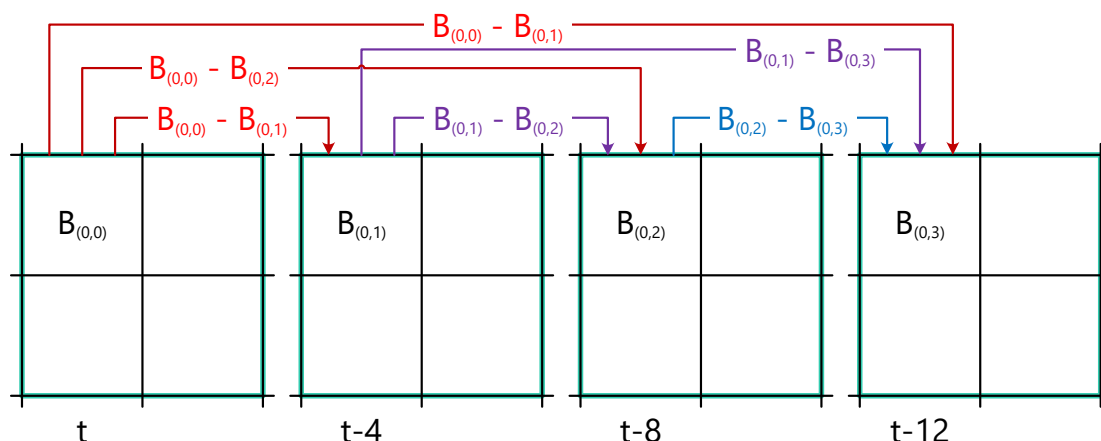
**Figure 5.12** — The static reconstruction creates the display frame by placing the samples from the four last received frames into the pixel  $(x, y)$  they originated from. © IEEE 2021 [43].

texture with a quarter of the full-resolution, which is directly passed onto the hardware encoder.

**Encoding** For the amortised encoding, four parallel encoder sessions are used, one for each sampling pattern. GTX, RTX and Titan cards from NVIDIA are restricted to only two encoder sessions running in parallel, therefore Quadro type cards are required, as they do not have an upper limit for parallel sessions. Neither AMD nor Intel limit the number of parallel encoder sessions. It would still be possible to only use two parallel encoder sessions, but this would increase the latency and the throughput since every frame would be encoded as a key frame. Therefore, the usage is restricted and requires four parallel encoder sessions. Each session uses its own ring buffer, containing the Direct3D 2D interop textures that are used for the output of the compute shader described in the previous paragraph, and works asynchronously.

**Decoding and Rendering** Decoding works similarly to the encoding, where there is one decoder session for each sample pattern. Once a pattern frame is decoded it is copied into the display buffer  $B$  that consists of four ring buffers, one for each pattern  $i$ , and each ring buffer contains four Direct3D 2D interop textures:  $B_{(i,j)} \mid (i, j) \in \{0, 1, 2, 3\} \times \{0, 1, 2, 3\}$ . Each incoming frame can be copied into the correct texture





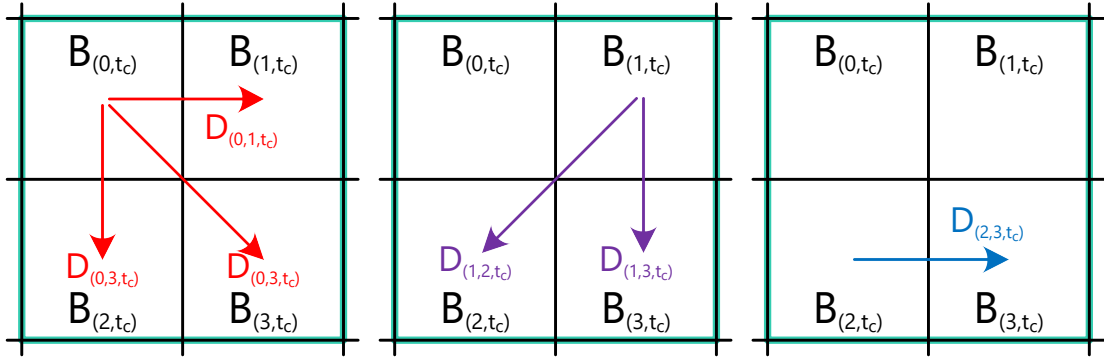
**Figure 5.13** — The absolute differences computed by the  $S(\mathbf{p})$  function for the first pattern. This is carried out for the other patterns as well for a total of 24 terms that are added up. The resulting sum indicates the change of the colour values over the last four frames. © IEEE 2021 [43].

using the number of the frame  $t$  and the definition that the first frame uses pattern 0:

$$i = (t/4) \bmod 4, j = t \bmod 4. \quad (5.1)$$

After the initial buffering, i.e. after more than 16 decoded frames, the most recent four frames for each pattern are available in the ring buffer. The display frame, which has four times the resolution of the decoded pattern frames, is then reconstructed in the pixel shader. This happens either directly, by copying the samples from the latest frame of each pattern  $B_{(*,j)}$  into the corresponding display pixel  $\mathbf{q} = (x, y)$  of the display frame (cf. Figure 5.12), or by only copying the samples of the latest decoded frame and interpolating the remaining display pixels based on these samples.

For every pixel of the display frame the index  $idx = (y \bmod 2) \cdot 2 + (x \bmod 2)$  of the pattern whose sample would be used by the static reconstruction is computed. Then the position  $\mathbf{p} = (x_p, y_p) = (x/2, y/2)$  of the sample is computed, which is the same for all incoming textures regardless of the pattern. This allows checking the colour change, of the samples, over the last four frames and detect significant changes. Using just two frames for this check makes it susceptible to repetition or fast movement, therefore a history of four frames per pattern is used to improve the test. Overall, this is a trade-off between quality and memory. Using additional frames improves the quality of this test but in turn increases the memory cost. The colour change  $V(\mathbf{p}) \in \mathbb{R}^3$  is computed for the three colour channels of the NV12 colour format, and compared to a given threshold value  $\epsilon = 0.012$ . The threshold value is required since the resulting colour of the lossy



**Figure 5.14** — The absolute differences computed by the  $F(\mathbf{p})$  function for the first time  $t$ . For the other times  $t - 4$ ,  $t - 8$  and  $t - 12$  the same differences are computed and then each is subtracted from its respective term at time  $t$ . This leads to 18 terms that are added up and the resulting sum indicates if the colour values in the neighbourhood have changed over the last four frames. © IEEE 2021 [43].

encoding and decoding process is subject to a temporal variation slightly below  $\epsilon$  even if the input colour is unchanged.

$$V(\mathbf{p}) = \frac{S(\mathbf{p}) + F(\mathbf{p})}{42} \quad (5.2)$$

$S(\mathbf{p})$  is the sum of absolute differences for a pixel for all unique combinations  $(j, k)$  of the last four frames (see Figure 5.13) per pattern  $i$ , summed up over all four patterns:

$$S(\mathbf{p}) = \sum_{i=0}^3 \sum_{j=0}^2 \sum_{k=j+1}^3 |B_{(i,j)}(\mathbf{p}) - B_{(i,k)}(\mathbf{p})| \quad (5.3)$$

This indicates to what extent the colour value of the sample has changed between the last four frames of the same pattern.

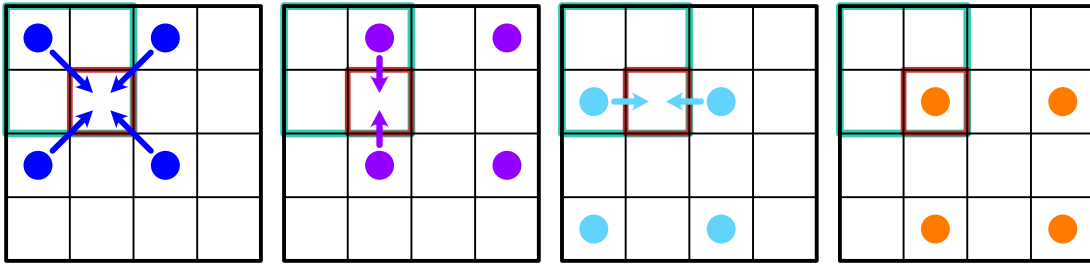
The second term  $F(\mathbf{p})$  computes how the absolute difference for a pair of patterns  $(i, i')$

$$D_{(i,i',t)} = |B_{(i,t)} - B_{(i',t)}| \quad (5.4)$$

has changed in the three previous frames with respect to the current frame  $t_c$ . It then adds up these for all unique pixel pattern combinations (see Figure 5.14):

$$F(\mathbf{p}) = \sum_{i=0}^2 \sum_{i'=i+1}^3 \sum_{k=1}^3 |D_{(i,i',t_c)} - D_{(i,i',t_p)}| \quad (5.5)$$

with  $t_p = t_c - k + 4 \bmod 4$



**Figure 5.15** – Where the colour has changed more than the threshold only the last decoded pattern frame is used to reconstruct the full resolution display frame. Depending on the pattern, the highlighted pixel (red) is either interpolated from adjacent samples as indicated by the arrows, or simply copies the colour value if it happens to line up with a sample. © IEEE 2021 [43].

where  $t_p$  refers to a previous frame, clamped to the valid ring buffer indices. Note that  $V$ ,  $S$ , and  $F$  depend on frame (or time) implicitly: they all use the complete set of ring buffers (all patterns and history). Proper storage is ensured when new frame data arrives, such that the most recent  $4 \times 4$  frames are always available and in sequential order (see Equation 5.1).

$V(\mathbf{p})$  indicates how stable the colour values have been across all frames currently in the ring buffer. The first term,  $S(\mathbf{p})$ , on its own fails to detect fast, especially periodic, movement, but is good at detecting slow movement. Therefore, the second term,  $F(\mathbf{p})$ , was introduced which is better suited to detect such movement, but not as good at detecting slow movement. The combination of both allows the accurate detection of movement and uses the best suited reconstruction method for the current pixel  $\mathbf{p}$ .  $S(\mathbf{p}) + F(\mathbf{p})$  ultimately contains a sum of 42 difference terms, which is averaged.

$V(\mathbf{p})$  is then compared to the threshold value and if it is smaller than the threshold, the static reconstruction is used (cf. Figure 5.12), i.e the colour value at sample  $(\mathbf{p})$  of pattern  $idx$  is copied into the display texture at pixel  $\mathbf{q}$ . If the change is bigger than the threshold only the last decoded pattern frame is used to compute the display frame. Since the decoded pattern frames each only contain image information for a quarter of all display pixels, the missing information is reconstructed using bilinear interpolation from the available pixels. As shown in Figure 5.15, this can be simplified to four cases which can be considered depending on the location of the currently computed display pixel  $\mathbf{q}$  in relation to the sample values of the last used pattern. If the pixel is located at the center of four adjacent samples of the pattern, all four values are used. If the pixel is located in-between just two directly adjacent samples, the interpolation is performed either horizontally or vertically between these two samples (the other values

are cancelled out). Finally, if the current pixel  $q$  directly lines up with the location ( $p$ ) of a sample from the last decoded pattern, the value can be copied directly.

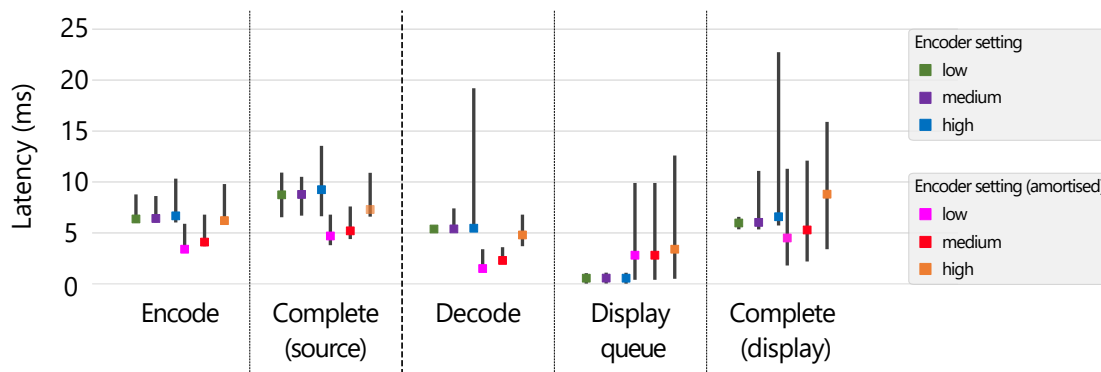
While the display frame is being reconstructed, the information required to do so is locked. Therefore, no new decoded pattern frames are added to the display buffer. This guarantees that a tearing free image is produced. The display of the last rendered frame is synchronised across all nodes that display it by either NVIDIA's Quadro Sync technology [110], if available, or by an MPI barrier as described in subsection 4.2.9.

### 5.2.3 Results and Discussion

For the quantitative evaluation of the amortised encoding the same setup, as described in section 4.3, was used to measure the latency and throughput in different scenarios. For the evaluation the four molecular dynamics visualisations, also described in section 4.3, were used. The resulting latency and throughput measurements are compared against the system described in chapter 4 in order to answer the questions: The focus of the evaluation was on the following questions: Does the amortised encoding have a negative effect on the encoding and decoding latency? By how much can the required bandwidth be reduced when using amortised encoding?

**Test Scenarios** In order to make the tests comparable to the baseline system the same settings were used: the low setting uses a quantisation value of 51, the medium setting a value of 31 and the high setting a value of 11. Both, the normal and the amortised encoding, were tested using each of the four molecular dynamics visualisations and each visualisation was tested once for each encoder setting.

**Test Results** Figure 5.16 depicts the measured latencies, in milliseconds, for the steps in the encoding and decoding pipeline that differ the most between the encoding approaches. Since they are similar for all four visualisations, only the latencies for the *crystal* dataset are shown. The plotted values depict the worst case measured across all nodes, with the variance between the nodes being about 1 ms throughout. The median values for the different configurations are represented by the coloured blocks, while the grey lines indicate the minimum and maximum values. Since their combined impact is well below 1 ms, due to the fact that the test was performed in a local area network, the latencies for the network and the MPI communication are not shown individually. Furthermore, the latencies for the frame conversion, the reordering as well as the frame assembly are not shown since they are equal for the normal encoding and the amortised encoding. However, they are included in the *Complete (Source)* and *Complete (Display)* columns. *Complete (Source)* contains the complete duration that covers the time from capturing a frame to sending it. *Complete (Display)* contains the complete duration from receiving a frame to displaying it. The *Display queue* column shows the duration

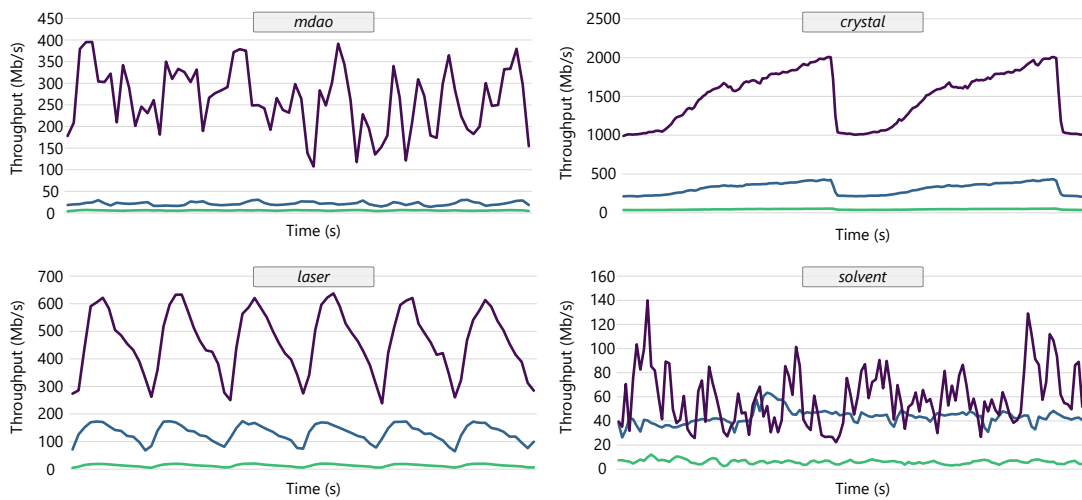


**Figure 5.16** — The measured latencies, in milliseconds, for the steps of the encoding and decoding pipeline that differ the most between the amortised and the normal encoding. Similar to Figure 4.10 the plot shows the median values (coloured block) as well as the minimum and maximum latencies, indicated by the grey lines. While the amortised encoding reduces the encoding and decoding latency, it increases the display queue latency. This results in a lower end-to-end latency.

of copying the decoded frame into the display queue as well as the time spent in the queue until the frame is displayed.

For the normal full-resolution encoding the major impact on the overall latency is the encoding, where the median is around 6 ms, while the maximum is around 8 ms for the low and medium settings. Using the high setting increases the median slightly to 7 ms and the maximum increases to 10 ms. For the display side most of the introduced latencies are negligible as the reordering, frame assembly, copying into the display queue and the duration in the display queue are all around or below 1 ms. The decoding takes 5 ms on average while the maximum latency rises up to 19 ms for the high setting.

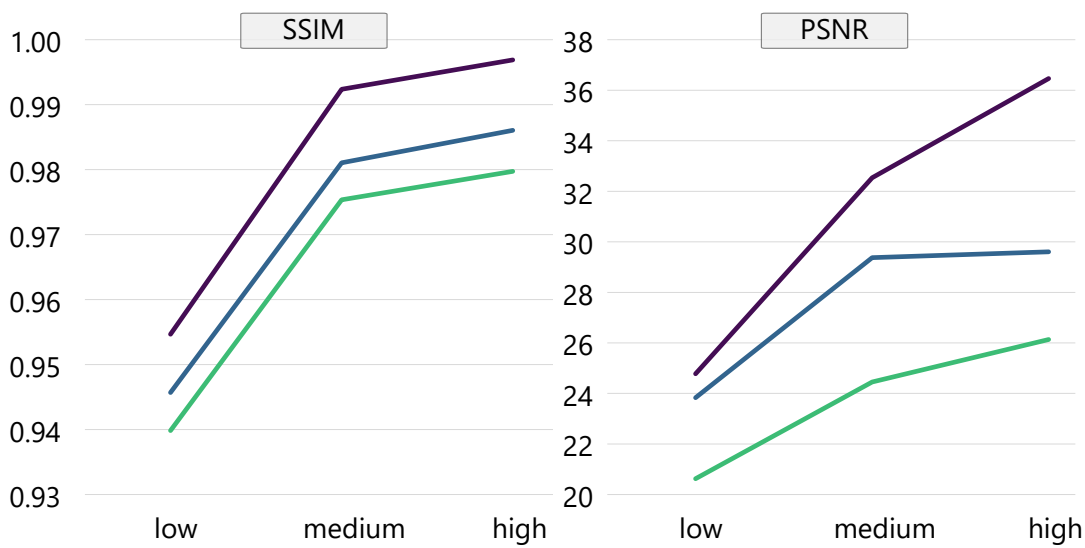
For the amortised encoding the major impact on the overall latency shifts from the encoding side to the display side. The median latency for the encoding of the *crystal* dataset is 3.4 ms for the low setting, 4.1 ms for the medium setting and 6.2 ms for the high setting and therefore significantly lower than for the full-resolution encoding using the low and medium setting. The maximum latency is 5.9 ms, 6.8 ms and 9.8 ms, which is again lower than for the normal encoding. Overall for the low and medium settings the minimum and median encoding latencies drop to about 56% compared to the normal encoding, while the maximum latency falls to around 74%. For the high encoder setting the encoding latencies show no significant changes. The same latency reduction was measured for the other datasets as well. The down-sampling, using the four patterns, does not change the amount of time it takes to convert the frame



**Figure 5.17** — The measured throughput for the four visualisations that were looped multiple times. The *mdao* (top left) uses at most 395 Mb/s for the high, 30 Mb/s for the medium and 6 Mb/s for the low setting. The *crystal* (top right) uses at most 2 Gb/s for the high, 420 Mb/s for the medium and 54 Mb/s for the low setting. For the *laser* dataset the maximum measured throughput was 637 Mb/s while for the *solvent* dataset the maximum was 140 Mb/s. It can be clearly seen that for the first three datasets the high encoder setting (■) requires significantly more bandwidth than the medium setting (■), which in turn requires significantly more bandwidth than the low setting (■). The only exception is the *solvent* dataset where the high setting is closer to the medium setting, sometimes even lower.

compared to the normal encoding.

On the display side the reordering and frame assembly latencies have not changed when compared to the normal encoding. The decoding latency has fallen, similar to the encoding latency, from 5 ms to 1.5 ms (minimum and median) and 3.4 ms (maximum) for the low setting. For the medium setting it fell from 5 ms to 2.3 ms (minimum and median) and from 7.4 ms to 3.6 ms (maximum), while for the high setting the minimum and median latency is slightly below the 5 ms of the full-resolution encoding but the maximum latency is reduced from 19 ms to 6.8 ms. However, the time a frame is in the display ring buffer has increased significantly to 3.4 ms (median) and 12.6 ms (maximum) for the high setting, which is partly due to the fact that the time to copy the frame into the buffer has increased, since the the display ring buffer is not updated during the time it takes to reconstruct and render the frame. For the medium and low setting the impact is slightly lower with 2.8 ms (median) and 9.9 ms (maximum). Therefore, the decoded frame has to wait until it is copied. The copy operation then takes less than



**Figure 5.18** — The computed SSIM and PSNR values between the captured and reconstructed images for the three different cases of the amortised encoding. For a fully static image (■) the SSIM value is very close to 1, which would be a perfect reconstruction, but it decreases for hybrid (■) and completely non-static images (■). The same is true for the PSNR values. © IEEE 2021 [43].

1 ms, similar to the normal encoding. Again the measured latencies are very similar for all four datasets.

In addition to the latency the throughput for each of the four datasets was measured. The aggregated maximum measured throughput of all nodes for the amortised encoding can be seen in Figure 5.17. The amortised encoding should offer a lower throughput since each encoded frame has a quarter of the resolution used for the normal encoding, but this is not the case for all datasets. This is due to the fact that only every fourth frame is passed to the encoder, therefore increasing the difference between the frames, which is especially prevalent for datasets with very little change on a frame-to-frame basis. The normal encoding performs much better for these datasets since the difference between successive frames is very small, increasing the efficiency of the encoding. A prime example is the *mdao* dataset where the amortised encoding increases the maximum throughput for the high setting to 395 Mb/s, compared to 250 Mb/s for the normal full-resolution encoding. For the medium setting the throughput is similar to the full-resolution encoding and for the low setting the throughput is lower, with at most 6 Mb/s versus 8 Mb/s. For bigger changes from frame-to-frame, like in the *solvent* dataset, the amortised encoding reduces the throughput to 80%, 49% and 23% for the low,

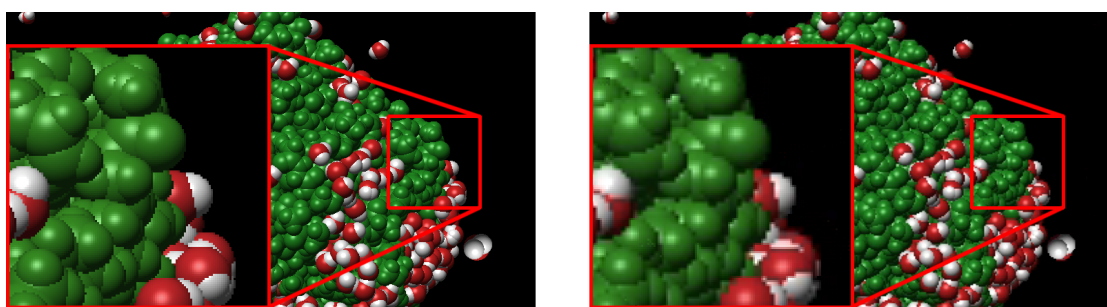


medium and high setting respectively. To summarise: in the worst case, the amortised encoding uses about 158% of the throughput of the normal encoding and in the best case 23%.

In order to evaluate the image quality of the amortised encoding the SSIM and the PSNR values between the captured and the reconstruction image were computed (cf. Figure 5.18), for three different cases. The first is a completely static image, the second is a hybrid case, where only parts of the image are static, and the third is a completely non-static image. For each case the three different encoder settings: low, medium and high were used. As expected for the static case the SSIM value is extremely close to 1, i.e. resulting in a perfect reconstruction, for all three encoder settings. In the hybrid case the SSIM value is also close to 1, although it is worse throughout, while for the non-static case the value falls again. For the PSNR values the same observation can be made, they are getting worse the fewer parts of the image are static.

The reduction for the encoding latency behaves as expected since the resolution is reduced to a quarter. There is only one minor exception, the high setting where the latency is nearly the same as for the normal encoding. The increase of the latency between the medium and high encoder setting is expected since it occurs for the normal encoding as well, but this increase is unexpectedly high. There is currently no explanation as to what causes this, but as the measurement could be reproduced multiple times it is not an outlier. Overall the median encoding latency is reduced to about 3 ms for the low and medium settings compared to 7 ms for the normal encoding. The median latency for the encoding using the high setting is similar when compared to the normal encoding with 6 ms versus 7 ms. The maximum latency is reduced by 74% for the low and medium settings while for the high setting it remains the same. The median end-to-end latency is reduced from 15 ms to 10 ms for the low and medium setting, while the maximum is reduced from 21 ms to 18 ms and 19 ms respectively. For the high setting the median end-to-end latency stays roughly the same with 16 ms while the maximum drops from 36 ms to 26 ms. For the amortised encoding the copy into the display ring buffers is a major contributor to the end-to-end latency. Copying the decoded frame from the decoder surface into the interop texture while it is used for rendering will lead to undefined behaviour. Not waiting for the rendering to finish and copying immediately after a frame is decoded causes the playback to visibly stutter, showing previous frames or tearing. Therefore, the copy waits for the rendering to finish, which introduces a latency of up to 10 ms, for the low and medium setting, while the median is 3 ms. The maximum latency for the high setting is 12 ms.

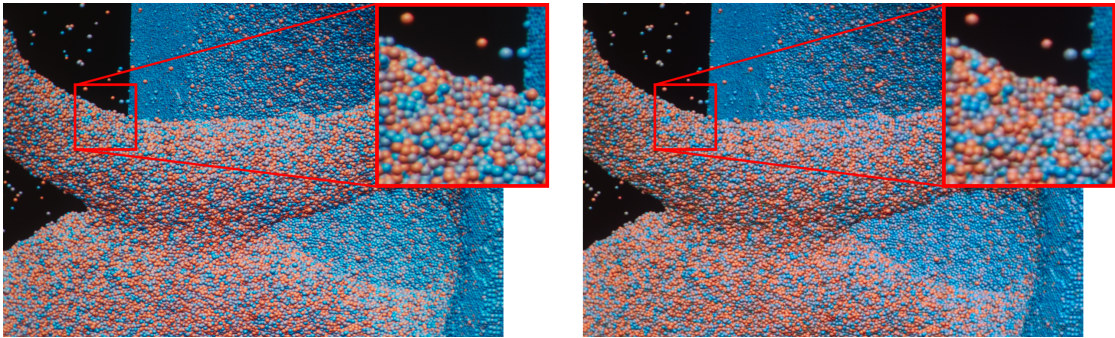
Originally the expectation was that, in addition to the reduced latency, the throughput would also be lower due to the reduced resolution. As can be seen in Figure 5.17 this is not always the case and depends on the visualisation. For two of the four datasets, *laser* and *crystal*, a similar, but slightly lower, throughput was measured for the amortised encoding. For the *mdao* dataset the throughput of the amortised encoding is higher,



**Figure 5.19** — Screenshot of the original rendering (left) and the reconstructed image (right), for a non-static region and a resolution of  $2560 \times 1440$ , using the high setting. The zoomed in parts show the difference in the image quality for a zoom level of 300%. As expected the image quality of the original rendering is better but the difference is hardly noticeable during movement and for normal viewing distances, especially on large displays. © IEEE 2021 [43].

using the high setting, and for the final dataset, *solvent*, the throughput is significantly lower. The difference lies in how big the change is between successive frames. If the change is small the encoding is extremely efficient. If the change is larger the encoding becomes less efficient. Since the amortised encoding only encodes every fourth frame for every pattern, the time difference between the frames is greater than for the normal encoding. This is the case for the *mdao* dataset, which was chosen for exactly that reason. The change between two successive frames is extremely small and therefore the normal, full-resolution, encoding is very efficient. The amortised encoding on the other hand is not, since the change between four frames is larger. In addition to this, the neighbouring pixels are not direct neighbours in the full-resolution frame, reducing the encoding efficiency further. This is due to the fact that in the full-resolution encoding the pixels of a macroblock are direct neighbours with mostly similar colour values, while in the sub-sampled frame the pixels of a  $16 \times 16$  macroblock come from a  $32 \times 32$  region in the full-resolution frame, which can increase the colour difference of those pixels.

As can be seen in Figure 5.19 and Figure 5.18, the image quality decreases only slightly in non-static regions. The two screenshots were captured on a machine that rendered the *mdao* dataset on a screen using a resolution of  $2560 \times 1440$  and shows the difference in the image quality between the original image on the left and the reconstructed image on the right. The reconstruction uses only one pattern as everything except for the black background is non-static, therefore the quality is lower, but this is barely noticeable at a normal viewing distance and while the visualisation is running. On the large display users have to be extremely close to the display to notice the difference and from further



**Figure 5.20** — Photos of the original rendering (left) and the reconstructed image (right), for a resolution of  $10800 \times 4096$ , using the medium setting. For the static reconstruction there is little difference to the original image since we use all four patterns. Therefore, any difference between the image is introduced by the encoding, i.e the low setting reduces the image quality significantly more than the high setting. © IEEE 2021 [43].

away the difference is no longer noticeable. For the static reconstruction there is no difference between the original and the reconstructed image, which can be seen on the two photos in Figure 5.20. Since all four patterns are used to reconstruct the image, any difference depends on the encoder setting, which is the same as for the normal, full-resolution, encoding.

As discussed in the beginning of this section, the amortised encoding faced the same major challenge as the amortised rendering: the temporal reprojection of previously rendered frames to compensate for motion in dynamic scenes. For synthetic rendering the complete scene information is available, such as camera positions and world space coordinates of rendered pixels. This information is not available after the encoding and therefore different techniques for the reconstruction step were explored. The first idea was to use the average of all patterns for non-static pixels but this resulted in visible ghosting. Even the weighted average, i.e using a higher weight for the latest pattern, and decreasing weights for older patterns did not resolve this ghosting. The next idea was to use the motion estimation, which is part of the H.264 codec, to extract the motion vectors and use them for the reconstruction to solve the temporal reprojection. Since none of the SDKs (Video Codec SDK [111], Advanced Media Framework SDK [7] and oneVPL [71]) offers a way to extract the motion vectors from encoded frames, another encoding pass was performed. NVIDIA’s encoder offers a motion estimation only mode that allows the extraction of motion vectors between two successive frames at the cost of an additional encoding step. This increased the encoding latency slightly but added an extreme overhead of 350 Mb/s to the throughput in order to transmit the motion vectors, making this not a viable solution. The last option that was looked at, was to compute

the optical flow for the decoded frames but this would nearly double the latency on the client side or require Ampere or Turing GPUs, which were not available for the large display. Therefore, the current reconstruction uses the interpolation approach for non-static regions.

### 5.3 Summary and Conclusion

Both approaches discussed in this chapter, reduce the required bandwidth needed for remote visualisation on large high-resolution displays. While the adaptive encoding increases the latency on the server side, the amortised encoding reduces the latency significantly. The adaptive encoding dynamically adapts the quantisation value for each macroblock based on the predictions of a CNN and an optimiser. The CNN predicts the image quality, based on SSIM, and the size of an encoded tile for three encoder settings. The optimiser then uses this information to determine the optimal settings, i.e. quantisation values for the macroblocks inside each tile, to yield the best possible quality for a given bandwidth threshold. The amortised encoding uses four patterns to sub-sample captured frames to a quarter of the resolution. On the client side the full resolution display frame is then reconstructed based on the decoded frames for the patterns. This allows the required bandwidth to be reduced for certain visualisations while preserving the image quality for static regions and only slightly impacting the image quality for non-static regions. Therefore, both methods can be used to preserve the image quality while reducing the required bandwidth. However, if there is a hard bandwidth limitation the adaptive encoding should be used since it stays below the given bandwidth, if possible, which is not guaranteed to happen when using the amortised encoding. On the other hand, if a lower latency is required, the amortised encoding should be used, which has the additional advantage that the image quality is preserved better for static frames.



## USER-DRIVEN OPTIMISATIONS

### Parts of this chapter have been published in:

- F. Frieß, M. Braun, V. Bruder, S. Frey, G. Reina, and T. Ertl. Foveated Encoding for Large High-Resolution Displays. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–10, 2020. [44]

While the methods discussed in chapter 5 already reduce the required bandwidth of the system discussed in chapter 4, they do not take the user into account. In visualisation in particular, details are crucial in areas that are currently under investigation. In contrast, lower quality is sufficient in other areas that are mostly required for context. The aforementioned methods still waste some of the limited bandwidth on image areas outside of the user's region of interest. In this chapter, methods to reduce the required bandwidth while keeping the image quality as close to the original as possible are discussed. The focus of these methods is on optimisations to the encoding that require user input, e. g. eye tracking, such as changing the quantisation of macroblocks within the field of view or sub-sampling the frame using a stippling pattern. The presented methods make use of GPUs to accelerate the necessary computations in order to achieve interactive frame rates.

Section 6.2 describes a method that tracks the gaze of users and changes the quantisation of each macroblock based on this information (cf. Fig. 6.1). This reduces the required bandwidth for transmitting a full high-resolution display image. Using the visual acuity described in subsection 2.1.5 the quantisation of the macroblocks is increased, i.e. the quality is lowered, the further away from the gaze point they are. Therefore, the client



side tracks the gaze of each user and sends the information to the server side where the visual acuity fall-off is computed and transformed into a QP offset map that is then used by the encoder.

Section 6.3 describes a method that tracks the gaze of users and uses a stippled 2D Gaussian function to (cf. Figure 6.11) reduce the number of pixels. Additionally, the method changes the quantisation of the macroblocks based on the gaze, similar to the previously described foveated encoding. Using the visual acuity described in subsection 2.1.5 the quantisation of the macroblocks is increased, i.e. the quality is lowered, the further away from the gaze point they are. This reduces the required bandwidth for transmitting a full high-resolution display image. Therefore, the client side tracks the gaze of each user and sends the information to the server side where the visual acuity fall-off is computed and transformed into a QP offset map that is then used by the encoder. Additionally, the client reconstructs the frames based on Voronoi cells using a natural neighbour interpolation.

## 6.1 Tracking

The methods described in this chapter rely on the users' gaze, i.e. they need to know where each user is looking. In addition to the gaze point other information is required: for example, the position of the user, the distance to the tiled display and the view direction. In order to transmit the tracking data, three different data types were defined. The first only contains the gaze point and the distance to the display in metres, the second contains the distance, the position of the user and the view direction, while the third contains four points on the display that form a region. The second and third data type is used by the foveated encoding, described in section 6.2, and the first data type is used by the stippling-based encoding, described in section 6.3.

In order to obtain the tracking data for each user, two systems were developed. The first uses optical tracking and the second uses the eye tracking capabilities of the HoloLens 2. Both are explained in detail in the following sections.

### 6.1.1 Optical Tracking

For the optical tracking, devices equipped with reflective markers are used. Each device, referred to as *rigid body*, has a unique pattern of reflectors that is used as an identifier in the system. The tracking system used consists of two Prime 13 cameras and 22 S250e cameras from NaturalPoint OptiTrack and use their software Motive to stream the current position, a vector in  $\mathbb{R}^3$ , and orientation, a quaternion, of each rigid body to any machine on the same network via UDP. It is vital to calibrate the initial orientation of a rigid-body in reference to the display. Therefore, the position of the lower left corner of



the display in the tracking coordinate system is determined. In addition to the position the height and width of the display, which determine the up vector and right vector of the display, are set. To calibrate a rigid body, it needs to be placed inside the tracked area, pointing towards the display. The orientation of the rigid body, as seen by the tracking system, is saved automatically, so each rigid body only needs to be calibrated once.

In order to compute the intersection point  $\mathbf{s}$  on the display plane, the observer's viewing direction  $\mathbf{d}$  is first determined.

$$\mathbf{d} = \tilde{Q}_{rb} \cdot Q_{rb}^{-1} \cdot \begin{pmatrix} 0 \\ -\mathbf{n}_d \end{pmatrix} \quad (6.1)$$

Here, the quaternion  $\tilde{Q}_{rb}$  is used that represents the current orientation of the rigid body, the inverse quaternion  $Q_{rb}^{-1}$  of its calibrated neutral orientation and the normal vector  $\mathbf{n}_d \in \mathbb{R}^3$ . The normal of the display plane  $\mathbf{n}_d$  points towards the tracking area in front of the display wall. Using the First Intercept Theorem [57], the distance  $\delta$  between the position of the rigid-body and the intersection point on the display is then computed:

$$\delta = \frac{\mathbf{n}_d \cdot (\mathbf{p}_{rb} - \mathbf{o}_d)}{\mathbf{n}_d \cdot (-\hat{\mathbf{d}})}. \quad (6.2)$$

Here,  $\hat{\mathbf{d}}$  is the normalised vector  $\mathbf{d}$ ,  $\mathbf{o}_d \in \mathbb{R}^3$  is the physical origin of the display, for which lower left corner is used, and  $\mathbf{p}_{rb} \in \mathbb{R}^3$  is the current position of the rigid body. Using the distance  $\delta$  the intersection point  $\mathbf{s}$  is computed:

$$\mathbf{s} = (\mathbf{p}_{rb} + \delta \cdot \hat{\mathbf{d}}) - \mathbf{o}_d \in \mathbb{R}^3. \quad (6.3)$$

In order to deal with different resolutions at the two locations the intersection point is converted into relative screen-space coordinates:

$$\hat{\mathbf{s}}(x, y) \in \mathbb{R}^2 : x = \frac{(\hat{\mathbf{r}}_d \cdot \mathbf{s})}{w}, y = \frac{(\hat{\mathbf{u}}_d \cdot \mathbf{s})}{h}, \quad (6.4)$$

with  $x, y \in [0, 1]$ .

Here,  $h$  and  $w$  denote the height and width of the display,  $\hat{\mathbf{u}}_d$  the normalised vector in the up-direction of the display and  $\hat{\mathbf{r}}_d$  the normalised vector in the right-direction. The latter two are determined during the calibration step.

### 6.1.2 HoloLens 2

The gaze point on the tiled display can be determined by using the built-in eye tracking capabilities of the HoloLens 2. First the user sets three points: the first is placed in the

lower left corner of the display, while the second and third are used to define the right and the up vector of the coordinate system. Therefore, the second point is placed in the lower right corner of the display and the third in the upper left corner. All points are within the coordinate system of the HoloLens 2.

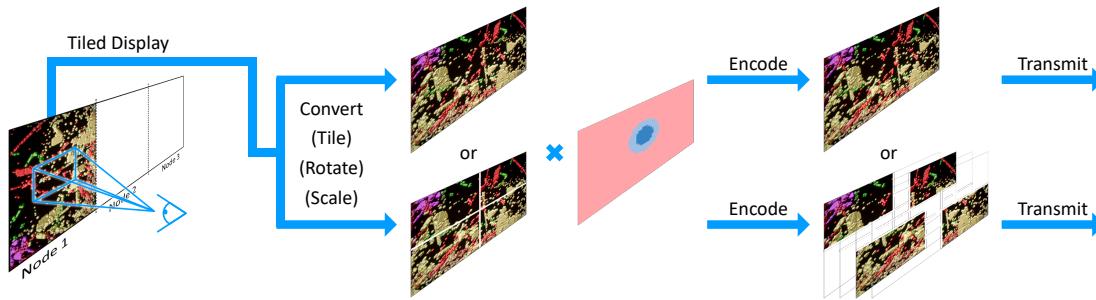
The Eye Tracking API of the HoloLens 2<sup>1</sup> provides access to a single ray, which consists of the gaze origin and direction, with 30 Hz. Both are transformed into the new coordinate system consisting of the three aforementioned points. From there, using the transformed gaze origin and direction, the intersection point on the tiled display can be calculated by computing the intersection between the gaze direction vector and the plane that represents the tiled display.

## 6.2 Foveated Encoding

Foveated video compression, achieved by changing the implementation of the video encoder, has been explored previously. Lee and Bovik [134] improved the efficiency of video processing by constructing several foveated video processing algorithms: foveation filtering (local bandwidth reduction), motion estimation, motion compensation, video rate control, and video postprocessing. Their approach led to a better computational efficiency by using a protocol between the encoder and the decoder. Wiedemann et al. [170] used eye-tracking to analyse the bitrates at the point of just noticeable distortion (JND) in order to evaluate the performance of an H.264 based foveated coding scheme. Chen and Guillemot [28] adapted the macroblock quantization adjustment in the H.264/advanced video coding by using a foveated model. This model enhances the spatial and temporal just-noticeable-distortion models in order to account for the relationship between visibility and eccentricity. For each macroblock the quantization parameter is optimized based on this model. Illahi et al. [69, 70] adapted the video encoder of a cloud-gaming application so that it changes the quality of the encoding based on the direction of gaze of the player. They adjust the quality parameter of each macroblock based on the current gaze position, increasing the quality in macroblocks the player looks at, and decreasing the quality in the remaining ones. Zare et al. [177] proposed using tiled based encoding in order to transmit wide-angle and high-resolution spherical panoramic video content to head-mounted displays. They store the video content in two different resolutions, divided into multiple tiles using the High Efficiency Video Coding (HEVC) standard. Based on the user's current viewport tiles are selected. For these tiles the highest captured resolution is transmitted while the remaining tiles are transmitted from the low-resolution version.

---

<sup>1</sup> <https://docs.microsoft.com/en-us/windows/mixed-reality/design/eye-tracking> (last accessed 22/04/2022)



**Figure 6.1** — Overview of the major parts of the pipeline from capturing a frame on the server side to sending it to the client side. Each captured frame is converted, and optionally rotated, downsampled or divided to fit the resolution requirements of the encoder. Based on the received foveated regions, the constant quality parameter for each macroblock of the encoder is adapted and the frame or the tiles are encoded. The client side decodes and displays the received frames or tiles. © IEEE 2020 [44].

In contrast to these techniques, our approach is not restricted to a single machine as it is an extension to the system discussed in chapter 4. Therefore, it is able to deal with different hardware set-ups used to build large high-resolution displays. Additionally, we adapt the size and compression of the foveated region based on the distance between the user and the display.

### 6.2.1 Method

The server side renders and captures the visualisation, and then carries out foveated encoding of the captured frames (cf. Figure 6.1). For this the quantisation of the macroblocks, used by the H.264 video codec, is changed based on their distance to the gaze points of users. The client side provides the respective gaze points, based on tracking the users, and displays the decoded frames.

The server side uses the following steps to produce the foveated encoding. This is computed in parallel on all nodes that render a part of the (potentially distributed) frame buffer. Firstly the last rendered frame is acquired as a texture. Since the encoder might expect a different colour format, a compute shader is used to perform the necessary conversion. This shader also handles the optional rotation, in 90° steps, in addition to the, also optional, downscaling or dividing into separate tiles of the texture in order to fulfil the resolution requirements of the encoder. It outputs either the converted, rotated and downsampled texture or multiple converted and rotated tiles. Initially all macroblocks are considered to be in the peripheral region. Then, based on the tracking data, i.e. foveated regions or contrast sensitivity, the new quantisation parameter for all

macroblocks is computed. All macroblocks that are not within a users gaze always use the highest possible quantisation parameter in order to reduce the required bandwidth. All encoded frames, or tiles, are sliced to fit UDP packages and forwarded to the client.

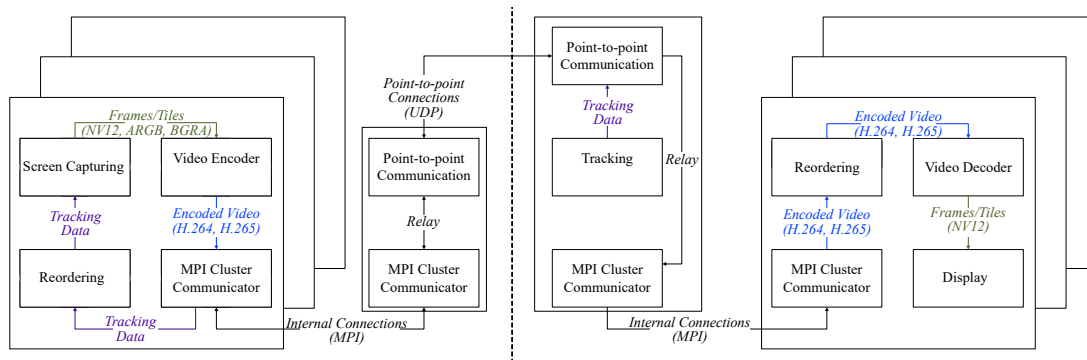
The client side uses the following steps to display the incoming encoded streams, which are again performed on all nodes, that render a part of the (potentially distributed) frame buffer, in parallel. All received UDP packages are reordered based on the timestamp and the, monotonically increasing, sequence number of the package. Then the (potentially sliced) frames are re-assembled and queued for decoding as well as display once they are complete. Incomplete frames are dropped after a user specified time has passed. Additionally, a single node on the client side computes the foveated regions by tracking the users and forwards these regions to the server side.

The following paragraphs provide additional details for the different steps of the foveated encoding and where it differs from the system described in chapter 4.

**Tracking Data** On the client side the tracking data needs to be acquired for every user, as described in section 6.1. For this three approaches are offered: the first uses an optical tracking system to track multiple users, the second uses the eye tracking capabilities of augmented reality devices, such as the HoloLens 2 and the third approach uses the position of the mouse cursor on the screen. All approaches deliver the tracking data to a single node on the client side. While the eye tracking delivers new gaze points with 30 Hz both, the optical tracking and the mouse update the data with 120 Hz. The mouse-based tracking was originally implemented for debugging but can also be used as an alternative if neither a tracking system nor eye tracking are available. Using the respective visual acuity fall-off for the tracking data, described in subsection 6.2.2, the data is mapped onto a QP offset map used by the encoder.

**Visual Acuity Fall-Off** As described in subsection 2.1.5 the ability of a person to recognize and distinguish small details is usually referred to as *visual acuity*. For each of the two tracking data types, described in section 6.1, used by the foveated encoding a different fall-off in visual acuity towards the periphery is defined. In the first method the visual acuity fall-off is modelled as a hyperbolic function. This model matches the density distribution of photoreceptors in the human macula and has been validated with low-level vision tasks [154, 22]. The second method uses a fall-off that is based on contrast sensitivity functions. The sensitivity of the human visual system at the foveal and peripheral region in the eye has been studied previously in terms of contrast sensitivity functions [128, 11].

**Encoding** The encoding and decoding has not changed compared to the baseline system except for the fact that the encoder uses a QP offset map to change the quantisation value of each macroblock based on the gaze of users. Otherwise the encoding uses



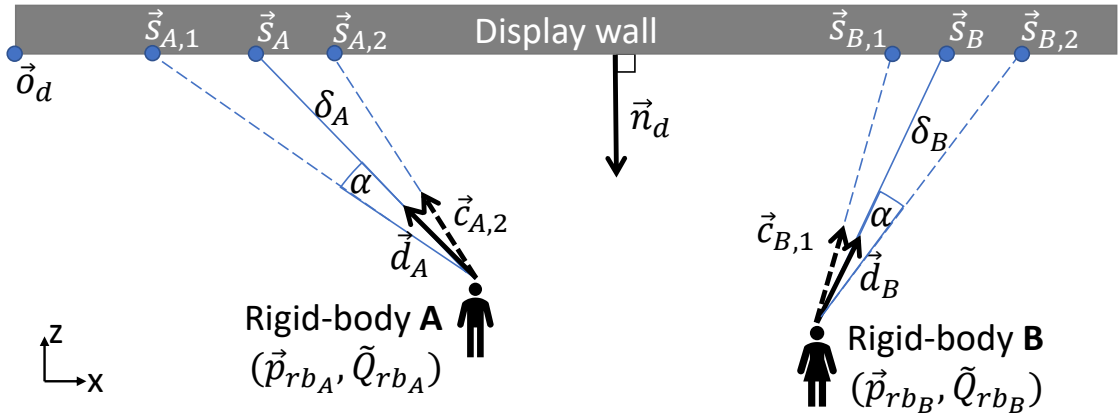
**Figure 6.2** — Overview of the system, which only shows the screen sharing path and the additional tracking data path. On the left is the server side and on the right the client side, separated by the dotted line. Depicted are the display nodes of the large displays and the streaming nodes, which transmit the data between the sides. For each node the components (see subsection 6.2.2) of the system and data flow between them are shown. Green-coloured connections represent raw images, while blue connections represent encoded images. Purple connections denote tracking data and the connections noted in black represent any of the above, i.e. data-agnostic communication links. © IEEE 2020 [44].

the constant quantisation mode, which sets the QP value of every macroblock to 51 prior to the application of the offset map. Furthermore the encoding does not limit the bitrate and uses an infinite GOP length with a regular intra refresh as error resilience measure, similar to the baseline system.

### 6.2.2 Implementation Details

The changes between the baseline system described in chapter 4 and the foveated encoding, i.e. the generation of the tracking data and the QP offset map, are also implemented using zero-copy and memory pooling on the CPU. However, the QP offset map is copied from the CPU to the GPU whenever a user moves. In the following paragraphs, the integration and interplay of the foveated encoding in the full system is described (cf. Figure 6.2).

**Tracking Data** The tracking data consists of either a rectangle (foveated region) or the distance and the position and the direction of view of the user. In the second case the distance can be computed using the respective intersection computation described in section 6.1. In the HoloLens case the position and view direction are given as well since they are required to compute the intersection. For the optical tracking system the



**Figure 6.3** — Schematic illustration of the optical tracking with two rigid-bodies A and B, depicted in the  $xz$ -plane. First, the intersection  $s$  of the view direction  $d$  with the display wall is calculated based on position and orientation of the rigid body. In a second step, the corner vectors  $c_i$  of the foveated region are determined using the angle  $\alpha$ , and are used to calculate the intersection points  $s_i$  with the display wall. © IEEE 2020 [44].

user position is known and the view direction can be computed using the intersection point and the position.

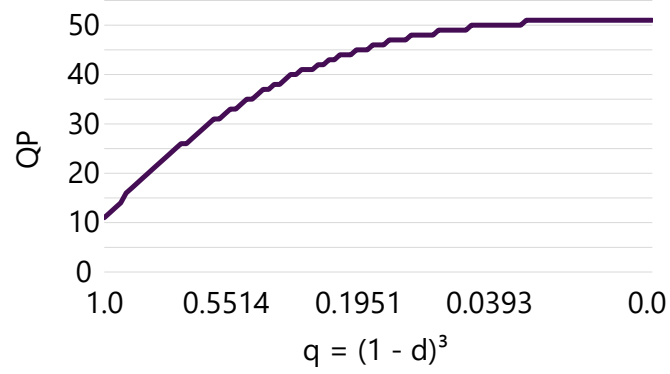
The foveated region is a rectangle that is spanned by four vectors  $c_i, i \in \{1, 2, 3, 4\}$ , starting from the position of the rigid body. A 2D illustration of the setup with two rigid bodies, including the relevant variables used in the calculation, is depicted in Figure 6.3. Therefore, the intersections with the display are computed as described in subsection 6.1.1. The horizontal angle  $\alpha$  and vertical angle  $\beta$  are used in order to compute the perspective projection. First, the current up vector  $u$  and right vector  $r$  of the rigid body are computed, based on the up- and right-vectors of the display as specified in the calibration step:

$$\mathbf{u} = \tilde{Q}_{rb} \cdot Q_{rb}^{-1} \cdot \begin{pmatrix} 0 \\ \hat{\mathbf{u}}_d \end{pmatrix} \in \mathbb{R}^3$$

$$\mathbf{r} = \tilde{Q}_{rb} \cdot Q_{rb}^{-1} \cdot \begin{pmatrix} 0 \\ \hat{\mathbf{r}}_d \end{pmatrix} \in \mathbb{R}^3$$

Half of the horizontal and vertical extents of the rectangle in normalised viewing

**Figure 6.4** – An example QP value distribution for the interval  $QP_{max} = 11$   $QP_{min} = 51$  based on the normalised distance between the gaze point, the centre of the foveated region, and the border of the foveated region. As can be seen, the quantisation increases rapidly while close to the gaze point and slows down as the distance increases.



directions are computed as follows:

$$\delta_\alpha = \tan\left(\frac{\alpha \cdot \pi}{180}\right)$$

$$\delta_\beta = \tan\left(\frac{\beta \cdot \pi}{180}\right)$$

The vectors  $c_i$  spanning the rectangle can then be computed with the normalised vectors  $\hat{d}$ ,  $\hat{u}$  and  $\hat{r}$ :

$$c_i = \hat{d} \pm (\delta_\alpha \cdot \hat{r}) \pm (\delta_\beta \cdot \hat{u}) \in \mathbb{R}^3 \quad (6.5)$$

Using Equations 6.2, 6.3 and 6.4, the relative intersection coordinates,  $\hat{s}_i \in \mathbb{R}^2$ , of the vectors with the display can be computed. If all points are outside of the interval  $[0, 1]$ , their coordinates will be changed to inf, otherwise points will be clamped to the interval. Both angles,  $\alpha$  and  $\beta$ , were set to  $25^\circ$ , in order to get a very conservative estimate of the bounding box surrounding the macula of an average human. The average human macula size (the region in the retina containing fovea, parafovea, and perifovea) is typically below  $20^\circ$ .

**Visual Acuity Fall-Off** For the foveated region a cubic function is used to approximate the fall-off function to determine a quantisation factor  $q$ , based on the distance  $d$  to the centre of vision. The distance  $d$  is computed between the centre of the foveated region and the centre of the macroblock and then divided by the maximum length, i.e, the distance between the centre of the foveated region and the top left corner of the foveated region.

$$q = (1 - d)^3, \text{ with } d \in [0, 1] \quad (6.6)$$

This function is applied in the foveal regions that are determined conservatively by assuming a  $50^\circ$  field-of-view as described in the previous paragraph. The encoding uses



the lowest quality outside this region. Due to the discrete nature of the quantisation parameter  $QP$ , used by the H.264 video codec, that is applied for encoding inside the foveal region, Equation 6.6 is approximated with a piecewise constant function to determine the quality parameter at a specific position:

$$QP = QP_{min} - \text{round}(q \cdot (QP_{min} - QP_{max})) \quad (6.7)$$

This produces a smooth approximation of the visual acuity fall-off function, and results in barely perceptible visual impact in the periphery (cf. Figure 6.4).

For the second data type, which consists of the distance to the display as well as the user position and view direction, the fall-off is approximated using contrast sensitivity functions, based on the work by Sheikh et al. [147]. The first step is to compute the vector  $\mathbf{b} \in \mathbb{R}^3$  from the user position  $\mathbf{p} \in \mathbb{R}^3$  to the centre of each macroblock:

$$\mathbf{b} = \begin{pmatrix} \mathbf{c} \\ 0 \end{pmatrix} - \mathbf{p}$$

Since the centre of each macroblock is in pixel it needs to be converted to metres, therefore the x and y coordinate is multiplied by the size of a single pixel in meter, resulting in the vector  $\mathbf{c} \in \mathbb{R}^2$ . Then the length  $d$  of the vector  $\mathbf{b}$  is computed, this is the distance between the user and the macroblock. The next step is to compute the eccentricity  $e_r$ , i.e. the angle, in radian, between the view direction  $\mathbf{v}$  and the vector  $\mathbf{b}$ :

$$e_r = \arccos\left(\frac{\mathbf{v} \cdot \mathbf{b}}{\|\mathbf{v}\| * \|\mathbf{b}\|}\right)$$

Then the eccentricity  $e_r$  is converted to degree  $e_d$  and corrected for eye imprecision and converted back to radian. The correction subtracts  $0.5^\circ$  from  $e_d$  if  $e_d$  is larger than 0.5, otherwise  $e_d$  is set to zero. The next step is to compute the eye frequency  $f_c$  for the macroblock:

$$f_c = \frac{e2 * \log\left(\frac{1}{CT_0}\right)}{\alpha * (e_d + e2)}$$

With  $e2$ ,  $CT_0$  and  $\alpha$  being constant values which are set to  $2.3^\circ$ ,  $\frac{1}{64}$  and 0.106 respectively according to Sheikh et al. [147]. The next step is to compute the display frequency  $f_d$  for the macroblock based on the distance  $d$  between the user and the macroblock:

$$f_d = \frac{\pi * d}{360} * \frac{2}{\cos(2 * e_r) + 1}$$

The final step is to normalise the ratio between  $f_c$  and  $f_d$  and then compute the desired QP value for the macroblock:

$$QP = QP_{min} - \min\left(1, \frac{f_c}{f_d}\right) * (QP_{min} - QP_{max}) \quad (6.8)$$

For both fall-off approximations it is also possible to provide a remapping that assigns pre defined QP value to  $QP_{max}$  based on the distance between the user and the display. This allows the QP value interval to be changed, i.e. the difference between the macroblocks close to the gaze point and the macroblocks on the periphery, based on the distance.

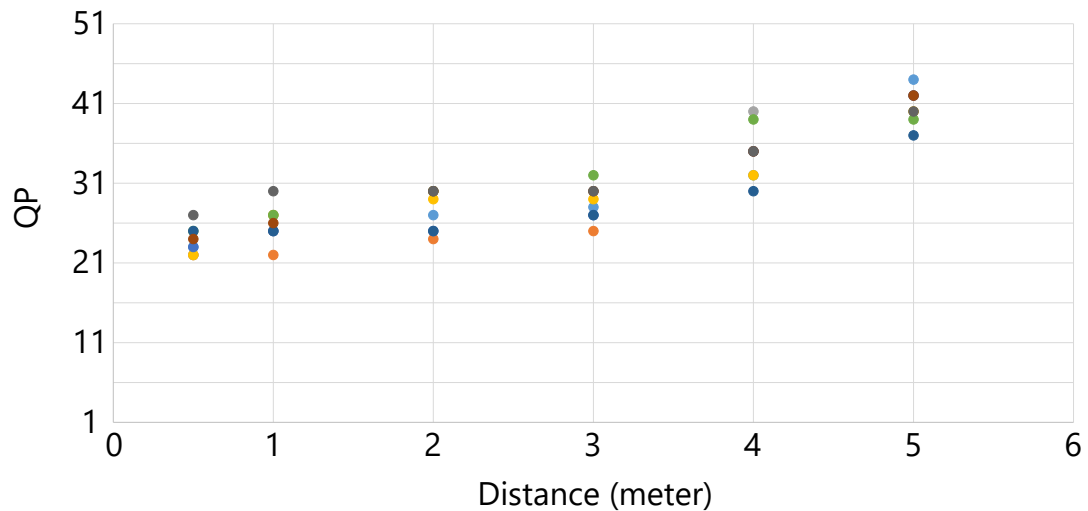
**Encoding** Similar to the CNN based optimisation described in section 5.1 a QP offset map is used that changes the quantisation parameter of each macroblock. Therefore only the NVIDIA Video Codec SDK (NVENC) can be used for the foveated encoding since the other SDKs (AMD and Intel) do not provide a function to alter the QP values of the macroblocks using this kind of map. The map is computed based on the type of the tracking data, which either consists of foveated regions, i.e. rectangles, or a list of distances, user positions and view directions. For both cases two maps are created in order to use double buffering, one of which is actively used by the encoder, while the second one is used to compute the new offset values. Once the new offset map is fully computed the roles of the maps are flipped. This avoids overwriting a map that is being used by the encoder at the same time.

For the foveated regions the intersections between each region and the macroblocks are computed. This determines which macroblocks are fully or partially inside each region. Then, for all macroblocks inside a region the distance  $d$  between the centre of the macroblock and the centre of the region is computed and scaled to the interval  $[0, 1]$ , see Equation 6.6. Based on the distance the desired quantisation parameter  $QP$  for the macroblock is computed using Equation 6.7. The rectangle and the centre of each macroblock are precomputed based on the overall size of the tiled display and the position of each machine. To compute the offset  $O$ , which is stored inside the map, the difference to the lowest quantisation parameter is computed:  $O = QP - 51$ .

For the second type of data the quantisation parameter  $QP$  for each macroblock is computed using the Equation 6.8 using the user position and view direction. The offset  $O$  is computed equally to the foveated region approach, While this method provides a better quality than the foveated region approach, it is computationally more expensive.

### 6.2.3 Results and Discussion

In order to answer the following questions a quantitative evaluation of the foveated encoding was performed: Does the foveated encoding have a negative effect on the encoding and decoding latency? Do different foveated intervals have an impact on the latency? By how much can the required bandwidth be reduced when using foveated encoding with different intervals as well as single and multiple users? For this the setup, as well as the molecular dynamics visualisations, described in section 4.3, were used to measure the latency and throughput in different scenarios. The results are compared against the system described in chapter 4.

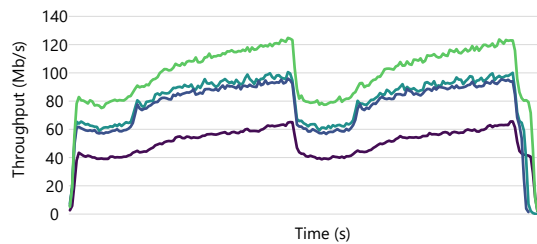


**Figure 6.5** — The QP values determined by the nine participants in the pilot study that were used to determine the minimum QP value for a certain distance. Each participant was shown the original rendering of a single frame of the *crystal* dataset, then the encoded frame using the QP value of 51 and then the original rendering again. This switching between rendering and encoded frame was carried out, with a decreasing QP value, until no difference was noticed. This process was repeated for the distances: 0.5, 1, 2, 3, 4 and 5 meters. The resulting QP values are close together, with two exceptions at one and four meter where the variance is 10 instead of the usual 5 to 6.

**Test Scenarios** The first preliminary test was performed to test the impact of the different data types used to compute the visual acuity fall-off for two different distances. The second preliminary test looked at the impact of the QP remapping on the throughput. After both tests the foveated encoding was tested using each of the four datasets described in section 4.3 by tracking one user and then two users who observed the visualisations simultaneously.

**Test Results** In order to determine the QP remapping values a pilot study with nine participants was performed. Each participant stood 0.5, 1, 2, 3, 4 and 5 meters away from the display and was shown the *crystal* dataset. The participants started five meters away from the display and were shown the original rendering. Then, switching back and forth between the encoded and the original rendering the QP value was decreased, starting with 51, until the participant noticed any differences or artefacts. Once the QP value for the distance was determined the participant moved closer to the display and the process was repeated. This resulted in the QP values shown in Figure 6.5. After all QP values were determined the foveated encoding with QP remapping was turned on, using the

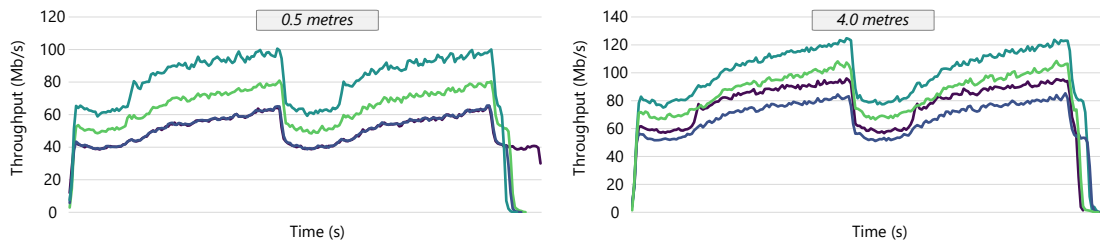
**Figure 6.6** — The measured throughput, in Mb/s, for the two different visual acuity fall-off functions for one user standing 0.5 and four meters away from the display looking at the same point. The rectangle-based foveated region uses less throughput, about 30 Mb/s, for both the near (■) and far(■) distance than the contrast-based foveated region (near (■), far(■)). This is due to the fact that the rectangle-based region is smaller and the fall-off is steeper.



determined QP values. Then the participants could walk freely in front of the display and were asked if any of them noticed any differences. This was repeated using the minimum, average and maximum QP values. Using the maximum values resulted in some users noticing the remapping, while using the minimum and average values none of them noticed any differences. Therefore, for the remapping tests the average QP values were used: (0.5m, 24), (1.0m, 26), (2.0m, 28), (3.0m, 29), (4.0m, 35), (5.0m, 41), resulting in the intervals [24, 51], [26, 51] and so on. For distances between the given points the QP value is interpolated linearly. Above five meters the the QP value is increasing linearly until 51 is reached after seven meters. And below 0.5 meters the same QP value (24) is used.

For the first preliminary test the *crystal* dataset was used to test the impact of the two visual acuity fall-off functions. The user focused on one point in the centre of the large display while standing four meters away from the screen and the visualisation was looped twice in order to measure the throughput. This was then repeated while the user stood 0.5 meters away from the display. Figure 6.6 shows the measured throughput, in Mb/s for the two functions. As can be seen the rectangle based fall-off required less bandwidth for both distances. This is due to the size of the foveated region, since the rectangle region is smaller, and the fall-off is steeper than for the contrast-based region. Overall the difference is 24 Mb/s for the four meter distance and 30 Mb/s for the 0.5 meter distance.

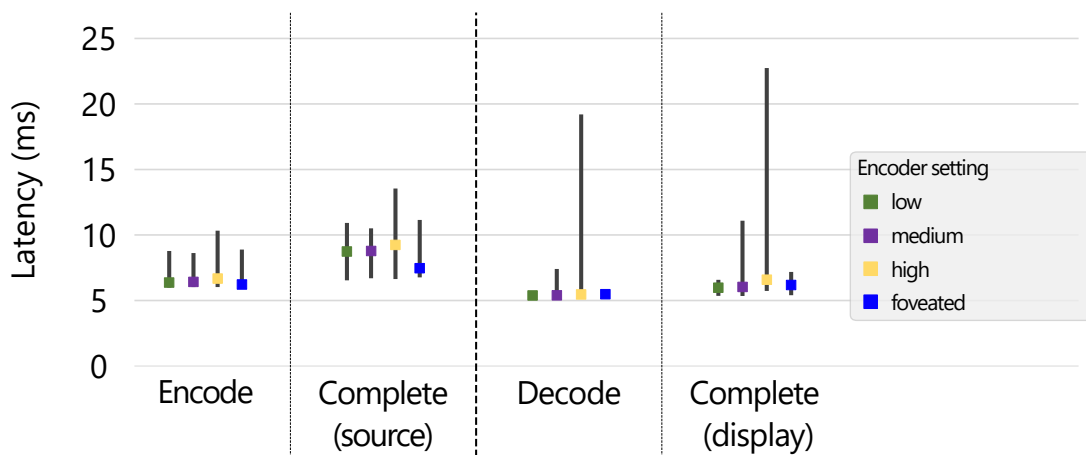
The second preliminary test looked at the impact of the QP remapping, discussed in subsection 6.2.2, on both visual acuity fall-off functions. This used the same setup as the first test, i.e. one user standing four and 0.5 meters away from the display, while streaming the *crystal* dataset. Figure 6.7 shows the measured throughput, in Mb/s for the two functions using remapping. For the rectangle-based foveated region the difference for the 0.5 meter distance is basically zero, so the remapping does not



**Figure 6.7** — The measured throughput for the two different visual acuity fall-off functions for one user standing 0.5 and four meters away from the display looking at the same point. For both functions the throughput for the near (■) and far(■) distance is shown with the throughput using the QP remapping (near (■), far(■)). For the rectangle based fall-off the remapping only reduces the throughput for the far distance and has no impact on the near distance, while for the contrast-based fall-off the throughput is reduced by 13 Mb/s for the 0.5 meter distance and by 16 Mb/s for the four meter distance.

reduce the throughput, while for the four meter distance the throughput is reduced by 19 Mb/s. For the contrast-based foveated region the remapping reduces the throughput by 13 Mb/s for the 0.5 meter distance and by 16 Mb/s for the four meter distance.

The latency changes slightly compared to the normal encoding, see Figure 6.8, which contains the latencies for the *crystal* dataset. For the other datasets the latencies are similar, with the maximum encoding latency of the *laser* dataset being the exception, similar to the baseline system. At no time during the tests did the observer look at one spot for longer than a couple of seconds. The observer attempted to visually cover all areas of the screen by walking in a random pattern and also moving closer and further away from the display in order to cover different-sized foveated regions. With the encoding and decoding latencies for one and two users being almost identical, only the latency values for the single user test were added. Both, the encoding and the decoding latency, is reduced to the level of the low encoding setting. For the foveated encoding the minimum and median latency is 6 ms, while the maximum latency is 8 ms and for the decoding the minimum and median latency is 5 ms while the maximum latency is 6 ms. Therefore the major impact on the overall latency is the encoding, while on the display side the decoding has the highest impact. When there is no foveated region present, the latency values do not change compared to the non-foveated encoding using the low settings. If there is a region present, the latency increases slightly, depending on the size of the region and on the interval. For the remapping the latency is slightly lower, while the user is four meter away from the display, since the QP value interval uses higher QP values. Overall, there is only very little difference in the encoding and

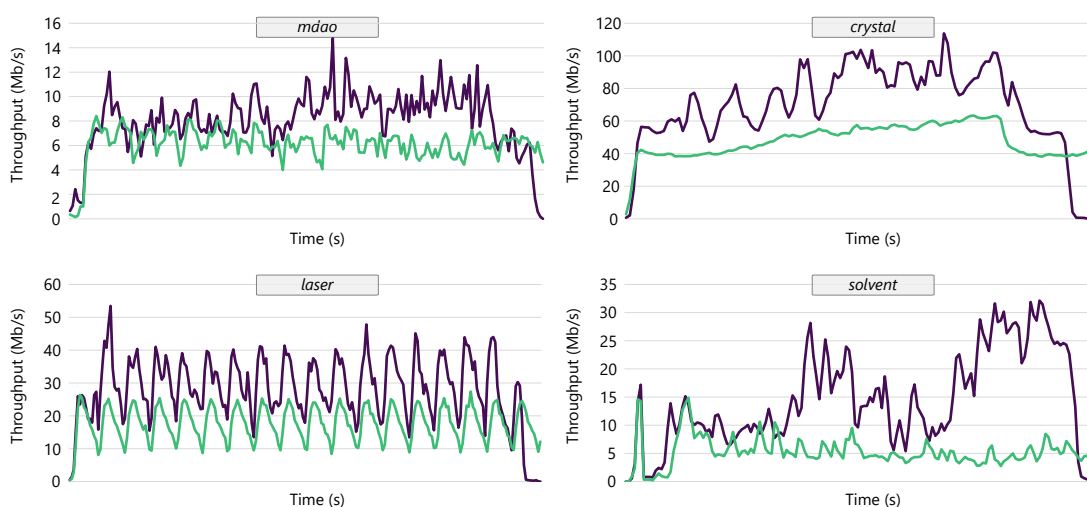


**Figure 6.8** — Overview of measured latencies for the encoding and decoding using the non-foveate and the foveated encoding. Similar to Figure 4.10 the plot shows the median values (coloured block) as well as the minimum and maximum latencies, indicated by the grey lines. Overall, there is only very little difference in the encoding and decoding latency between the foveated and the non-foveated case using the low setting. However, the maximum latency is drastically reduced compared to the medium setting and especially the high setting.

decoding latency between the foveated and the non-foveated case.

In addition to the latencies the throughput for each of the four datasets was measured. For these tests the QP remapping was used in combination with the contrast-based visual acuity fall-off function and similarly to the latency measurements the user walked in a random pattern in front of the display. The measured throughput can be seen in Figure 6.9. For the *mdao* dataset the maximum measured throughput is 15 Mb/s compared to 250 Mb/s for the high and 8 Mb/s for the low setting. The same can be observed for the other datasets as well, the measured throughput is similar to the non-foveated encoding using the low setting, with occasional peaks that increase the bandwidth. These peaks occur when the user stands further away from the screen and therefore the foveated region becomes larger. Tracking more users increases the throughput, depending on where both users looked at, but the average was below 80 Mb/s for the *crystal* dataset.

End-to-end latency across all scenarios was hardly noticeable in side-to-side comparison, as it was at most 15 ms and 12 ms on average. The major contribution to the overall latency, with up to 8 ms on the source side, comes from the encoding. For the foveated encoding the latency ranges between the low setting, when no foveated region is present, and the high setting, if the region covers the whole display. During the tests

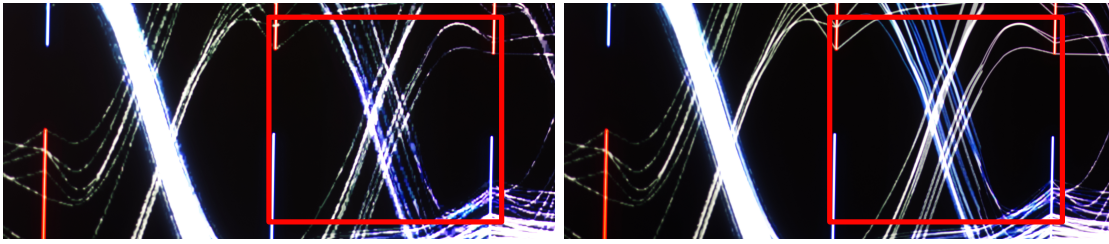


**Figure 6.9** — Aggregated measured throughput of all nodes for the four datasets, *mdao* (top left), *crystal* (top right), *laser* (bottom left) and *solvent* (bottom right). The throughput measured for the foveated encoding (■) is similar to the low encoder setting using the baseline system (■), with occasional peaks occurring when the user stood further away from the display.

the latter was at no time encountered, and the latency was on average around 6 ms and at most 8 ms. The latency for the acquisition, transmission and processing of the foveated regions was between 2 and 8 ms.

The measured throughput for the foveated encoding at no time exceeded 120 Mb/s, with the highest measured peak at 115 Mb/s, for the *crystal* dataset. The same can be said for multiple users, they did not increase the measured throughput by a significant amount. Therefore, the foveated encoding requires on average between 10 and 20 Mb/s more than the low setting, except for in situations when users want to get a complete overview of the visualisation, see the peaks in Figure 6.9. Similar to the latency, the upper and lower limit of the throughput are given by the interval of the quality parameter. The high encoding setting sets the upper limit at 2.5 Gb/s, and the low setting set the lower limit at 62 Mb/s, for the *crystal* dataset. By using the QP remapping the upper limit can be reduced further to fit the maximum available bandwidth. Although the pilot study needs to be extended to include more participants and more than one visualisation, it shows that there is little to no noticeable difference, for the molecular visualisation, between a QP value of 11 and 24, even if the user is closer than 0.5 metres to the display. All in all the foveated encoding with the QP remapping uses, on average, 14 percent of the measured throughput required for the non-foveated medium encoder settings (4





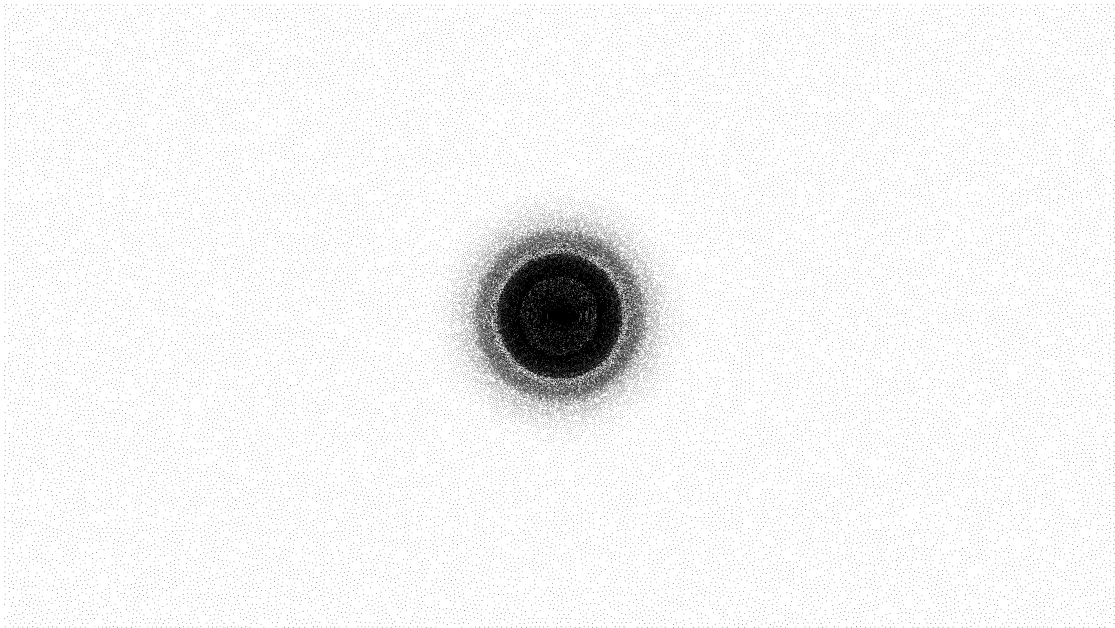
**Figure 6.10** — Two photos that show the difference between no foveated region (left) and the foveated region (right), using the QP interval  $[11, 51]$ , at the example of parallel coordinates. With the lowest quality parameter (51) the lines on the left photo are not clearly visible, sometimes not at all. The same is true outside of the foveated region, but the lines inside it are clearly visible. The two red rectangles cover part of the foveated region and highlight the differences. © IEEE 2020 [44].

percent compared to high encoder settings), while providing the same quality locally in the areas the users are focusing on.

As can be seen in Figure 6.10 using the example of a parallel coordinates visualisation, the foveated region visibly increases the quality locally. Fine lines that are hardly visible using the lowest encoding quality are clearly defined while using roughly the same throughput. Using both, the rectangle and the contrast-based fall-off it was not visually identifiable that the rest of the image was encoded with the lowest possible quality, since the extremely conservatively-sized foveated region perfectly covered the macula of an average human. However, for all tests the contrast-based fall-off was used, despite the fact that it requires more bandwidth, since it results in a better quality due to the fall-off being less steep. The high precision of the tracking system detected very small movements, which introduced a distracting flickering since the foveated region moved as well, changing the quality parameters for every frame. Therefore, a small threshold was introduced to filter out small movements and to eliminate the flickering. If the difference between the previous top left corner and the current top left corner of the look-at rectangle is smaller than one centimetre the rectangle is dropped. The same threshold is used for the contrast-based region as well. There the distance between the current look-at point and the previous look-at point is used.

### 6.3 Stippling-based Encoding

In order to reduce the information of a frame prior to encoding we used the same approach as Bruder et al. [23]. They applied a pre-computed sample map (cf. Figure 6.11) based on the visual acuity fall-off, using the Linde-Buzo-Gray algorithm [32, 49], in order



**Figure 6.11** — The approximation of the visual acuity fall-off using a 2D Gaussian function, which depends on the screen resolution, the screen size, the approximated viewing distance and the estimated photoreceptor distribution. This function is then stippled using the Linde-Buzo-Gray algorithm, resulting in sample positions which are shown as black dots.

to reduce the number of rays for Volume Rendering, which increased the performance. By reducing the number of samples prior to the encoding, which works similar to the foveated encoding, the required bandwidth is reduced further.

### 6.3.1 Method

The server side renders and captures the visualisation, and then carries out the stippling-based encoding of the captured frames. For this, a pre-computed sample map (cf. Figure 6.11) based on the visual acuity fall-off, using the Linde-Buzo-Gray algorithm [32, 49], is used to reduce the number of samples. This map is moved according to the users gaze. Then the quantisation of the macroblocks, used by the H.264 video codec, is changed based on their distance to the gaze points of users. The client side provides the respective gaze points, based on tracking the users, and displays the decoded frames after reconstructing the frames based on Voronoi cells using a natural neighbour interpolation.

The server side uses the following steps to produce the stippling-based encoding. Each

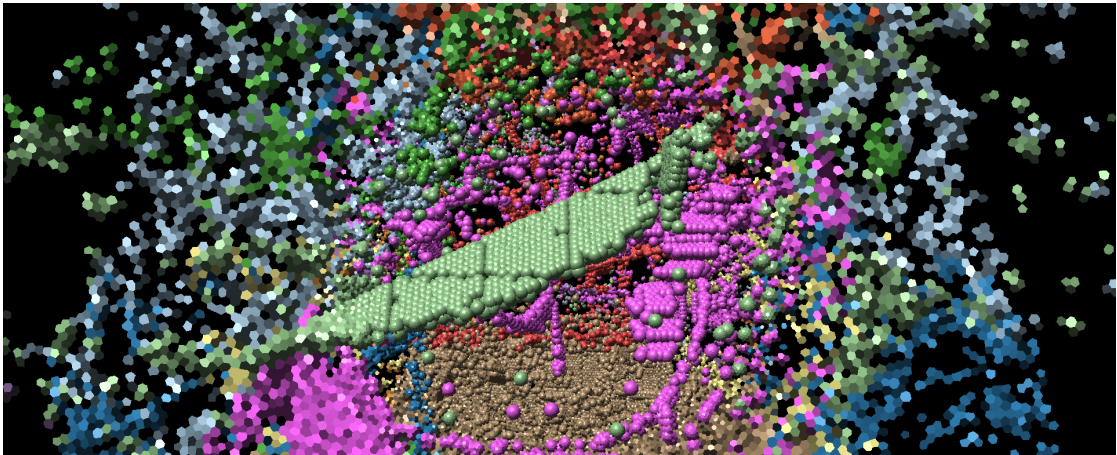
step is computed in parallel on all nodes that render a part of the (potentially distributed) frame buffer. Firstly, the last rendered frame is acquired as a texture and, since the encoder might expect a different colour format, a compute shader is used to perform the necessary conversion. This shader also handles the optional rotation, in 90° steps, in addition to applying the sample map. It outputs the converted and optionally rotated texture or multiple converted and rotated tiles. Initially all macroblocks are considered to be in the peripheral region. Then, based on the tracking data, i.e. the distance and the gaze point on the display, the new quantisation parameter for all macroblocks is computed. All macroblocks that are not within a user's gaze always use the highest possible quantisation parameter in order to reduce the required bandwidth. All encoded frames, or tiles, are sliced to fit UDP packages and forwarded to the client.

The client side uses the following steps to display the incoming encoded streams, which are again performed on all nodes, that render a part of the (potentially distributed) frame buffer, in parallel. All received UDP packages are reordered based on the timestamp and the, monotonically increasing, sequence number of the package. Then the (potentially sliced) frames are re-assembled and queued for decoding. Each decoded frame is then reconstructed using the natural-neighbour-based interpolation and copied into the display queue. Incomplete frames are dropped after a user-specified time has passed. Additionally, a single node on the client side computes the foveated regions by tracking the users and forwards these regions to the server side.

The following paragraphs provide additional details for the different additional steps of the stippling-based encoding as well as steps that are now different compared to the system described in chapter 4.

**Maps** In order to reduce the computation time of the natural-neighbour-based interpolation three maps are precomputed. The first (sample map) contains the sample positions used by the compute shader prior to encoding, the second (index map) contains the indices of all neighbouring Voronoi cells used by the natural-neighbour-based interpolation and the third (weight map) contains the weights. The index and weight map have twice the resolution of the display in order to accommodate for the gaze-dependent translation of the maps. Since users do not stand in front of a large display at a fixed position multiple maps of this kind have to be pre-computed, one for each required distance. Based on the distance of the user from the display, the closest sample map, index map and weight map is chosen for the encoding and reconstruction.

**Tracking Data** On the client side the tracking data needs to be acquired for every user, as described in section 6.1. For this, two approaches are offered: the first uses an optical tracking system to track multiple users while the second uses the eye tracking capabilities of augmented reality devices, such as the HoloLens 2. All approaches deliver the tracking data, i.e. the gaze point and the distance, to a single node on the client



**Figure 6.12** — Part of a frame of the *crystal* dataset after the frame conversion. As can be seen, the pixels further away from the centre of the image, which coincides with the gaze point, have the same colour, revealing the Voronoi cells. Towards the centre of the image the number of samples increases and more details of the captured frame are visible.

side. While the eye tracking delivers new gaze points with 30 Hz, the optical tracking updates the data with 120 Hz. Using the respective visual acuity fall-off for the tracking data, described in subsection 6.3.2, the data is mapped onto a QP offset map used by the encoder.

**Visual Acuity Fall-Off** As described in subsection 2.1.5 the ability of a person to recognize and distinguish small details is usually referred to as *visual acuity*. For the tracking data used by the stippling-based encoding, described in section 6.2, a fall-off in visual acuity towards the periphery is defined. The fall-off is approximated by a 2D Gaussian function that matches the density distribution of photoreceptors in the human macula and has been validated with low-level vision tasks [154, 22].

**Screen Capturing** Compared to the baseline system, the compute shader that converts the colour format as well as applying the optional rotation and downscaling or tiling, has changed significantly. While it still performs all of the aforementioned actions it now additionally applies the sample map. Each pixel copies the colour value from the last captured frame according to the closest sample, i.e. its closest natural neighbour. In theory only the sample positions need to be copied, but this would result in sparse images, for which the video encoding introduces problems, such as loss of colour and

colour bleeding. Therefore, the resulting frame contains only the colour values of the samples spread out to all pixels that are in the same Voronoi cell, see Figure 6.12.

**Encoding** The encoding and decoding has not changed compared to the baseline system except for the fact that the encoder uses a QP offset map to change the quantisation value of each macroblock based on the gaze of users. Apart from this one exception, the encoding uses the constant quantisation mode, which sets the QP value of every macroblock to 51 prior to the application of the offset map. Furthermore, the encoding does not limit the bitrate and uses an infinite GOP length with a regular intra refresh as an error resilience measure, similar to the baseline system.

**Decoding and Reconstruction** The decoding itself has not changed compared to the baseline system but the image is not displayed directly after decoding. Instead, the natural neighbour interpolation is used to determine the colour value of each pixel based on the colour values at the sample positions. Additionally, a temporal smoothing is used to reduce flickering in the peripheral regions. For this,  $n$  previous frames are used to average the colour value at each pixel. The display of the reconstructed frame is again unchanged compared to the baseline system.

### 6.3.2 Implementation Details

The implementation of builds on the baseline system described in chapter 4, as well as the foveated encoding, described in subsection 6.2.2. Since the encoding also uses the QP offset map and requires tracking data the implementation has not changed compared to the foveated encoding. However, the sample map, the index map and the weight map need to be copied from the CPU to the GPU. As this happens during the initialisation of the system the impact on the encoding performance is zero. Nevertheless, the compute shader has changed and now requires more data for each frame, an additional CPU to GPU copy is performed whenever a user moves. The same is true for the reconstruction after the decoding, which also adds a copy from the CPU to GPU. In the following paragraphs, the integration and interplay of the stippling-based encoding in the full system is described.

**Maps** The sample map, index map and weight map are computed using a modified Linde-Buzo-Gray algorithm [143], which was also used by Bruder et al. [23]. The index and weight map have twice the resolution of the display and therefore would require approximately 20 Gb of RAM or VRAM. Since multiple maps are required to use the stippling-based encoding, the size needs to be reduced. Each pixel has up to 16 neighbouring cells that are used for the interpolation, therefore the maps contain 16 values per pixel, not all of which are valid indices or weights, since not every pixel



has 16 neighbouring cells. Therefore, the size of the maps is reduced by removing the invalid indices and weights, reducing the combined size of each index and weight map pair to about 7 Gb. The index map contains the offset of each pixel, and after the offsets, the IDs of the samples that are used for the interpolation. Additionally, a reduced index map is created which contains only the closest neighbour for each pixel. This map is used by the compute shader that converts the colour in order to determine the sample that provides the colour value for each pixel.

**Tracking Data** The tracking data consists of the distance and the intersection of the view direction vector with the display on the client side. The  $x$  and  $y$  coordinates of the intersection are in the range of  $[0, 1]$  in order to work with different resolutions on the server and the client side. Using the optical tracking, this intersection can be computed using the algorithm described in section 6.1, while for the HoloLens the position and the direction of view can be used to compute the intersection point using the same formula. The distance is used to select the closest sample map and the intersection point is used to compute a rectangle that covers most of the samples that are close to the gaze point. Using the radius  $r = 3\sigma$  to determine the top left and bottom right corner creates a rectangle that covers over 90% of the samples.

**Visual Acuity Fall-Off** Since the stippling-based encoding uses a foveated rectangle, the fall-off approximation of the foveated encoding (cf. Equation 6.6 and Equation 6.7) is used. The distance  $d$  is computed between the intersection point and the centre of each macroblock and divided by the radius  $r$ , so that  $d$  is in the range of  $[0, 1]$ . Then the QP value is computed according to Equation 6.7 and the remapping that changes  $QP_{max}$  based on the distance between the user and the display.

**Screen Capturing** The original system uses the Desktop Duplication API [105] to gain fast access to the full desktop as a texture in the BGRA colour format. This texture is then converted to whichever colour format the encoder requires, and optionally rotated, downscaled or tiled to fit the maximum resolution of the encoder using a compute shader. For the stippling-based encoding a new compute shader was added that continues to perform the same actions but additionally applies the sample map. For each pixel  $(x_p, y_p)$  the index is computed and used to look up the ID of the sample in the reduced index map. The colour value of the pixel  $(x_p, y_p)$  is then acquired from the captured frame by using the optionally rotated position of the sample  $(x_s, y_s)$ , shifted by the gaze point, as the texture coordinates. This results in a new texture only containing the colour values of the samples spread out in each Voronoi cell, which is directly passed onto the hardware encoder.

**Encoding** The encoding works in a similar way to the foveated encoding, i.e. a QP offset map is used to change the quantisation parameter of each macroblock. Therefore, only the NVIDIA Video Codec SDK (NVENC) can be used for the stippling-based encoding. Double buffering is used to avoid concurrent reads and writes, and the offset values are computed as described in subsection 6.2.2.

**Decoding and Reconstruction** The reconstruction uses the decoded image, which contains the colour values of the samples, as well as the last  $n = 8$  reconstructed frames to create the next frame. The first step in the reconstruction is to perform the natural-neighbour-based interpolation. For this a new Voronoi cell is inserted at a given point  $(x, y)$  into the existing Voronoi tessellation. The estimate  $G$  of the new point is then computed by using the areas  $A$  of the intersections with neighbouring cells in relation to the total area of the new cell as weighting factors for the interpolation:

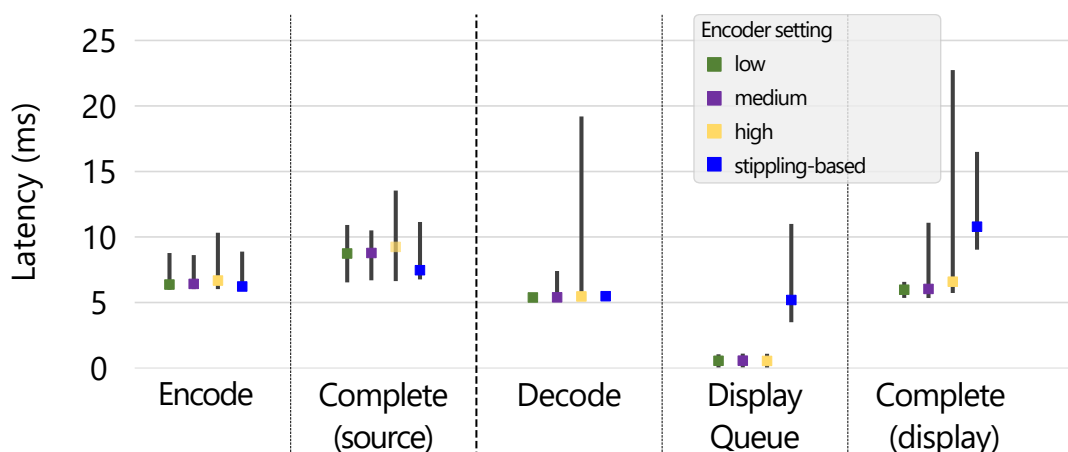
$$G(x, y) = \sum_{i=0}^k \frac{A(S_i \cap N)}{A(N)} * f(x_i, y_i) \quad (6.9)$$

$A(N)$  denotes the area of the new cell  $N$  and  $f(x_i, y_i)$  are the known values at the  $k$  neighbouring cells  $S_i$ . As mentioned previously, these values are precomputed and can be accessed using the index and weight map. The reconstruction is computed using a compute shader which determines the index of each pixel and then acquires the sample IDs and weights from the index and weight map respectively. The sample coordinates are again translated using the gaze point and then used to look up the colour value of the sample in the decoded image. All colour values are multiplied by their respective weight and added together.

The natural neighbour interpolation provides a precise and smooth reconstruction. However, the sparse sampling introduces aliasing artefacts at hard transitions, such as edges. Furthermore, under-sampling artefacts can occur, especially near fine structures, due to the lower sampling density in the peripheral vision. Since the peripheral vision is particularly sensitive to contrast changes and movement, additional temporal smoothing is required by averaging between previous frames. In addition to reducing artefacts the smoothing also provides a form of anti-aliasing and it hides the transition from blurry to sharp during rapid eye movement. To avoid introducing a motion blur effect, the temporal smoothing is only applied when the scene is static.

After the frame is reconstructed it is queued into the display queue and then displayed. In contrast to the baseline system, the frame remains in the display queue until it is no longer required for the temporal smoothing. This implies a larger display queue as each frame remains in the queue for longer.



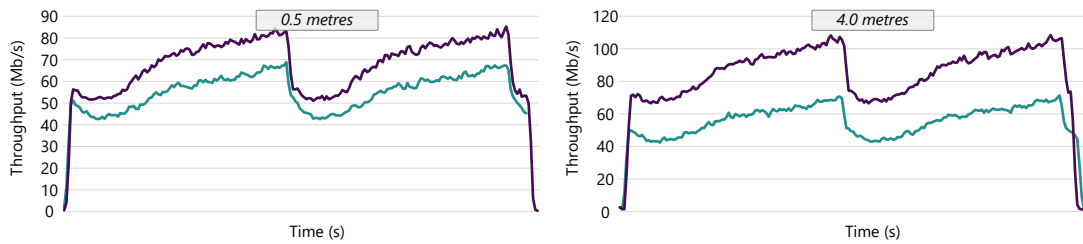


**Figure 6.13** — The measured latencies for the encoding and decoding steps using the stippling-based encoding and the normal encoding. Similar to Figure 4.10 the plot shows the median values (coloured block) as well as the minimum and maximum latencies, indicated by the grey lines. Overall, there is only very little difference in the encoding and decoding latency between the stippling-based encoding and the normal encoding using the low setting. However, using the normal and high settings increase the latency for the normal encoding and decoding. The other major difference is the display queue latency that contains the time required to reconstruct the frame.

### 6.3.3 Results and Discussion

A quantitative evaluation of the approach was performed by measuring the latency and throughput required to share the content of a tiled display in a local area network setting, as described in section 4.3. The throughput and latency were measured using four different molecular dynamics visualisations, also described in section 4.3 and compared to the system described in chapter 4 in order to answer the following questions: Does the stippling-based encoding have a negative effect on the latency? To what extent can the required bandwidth be reduced when using stippling-based encoding with single and multiple users?

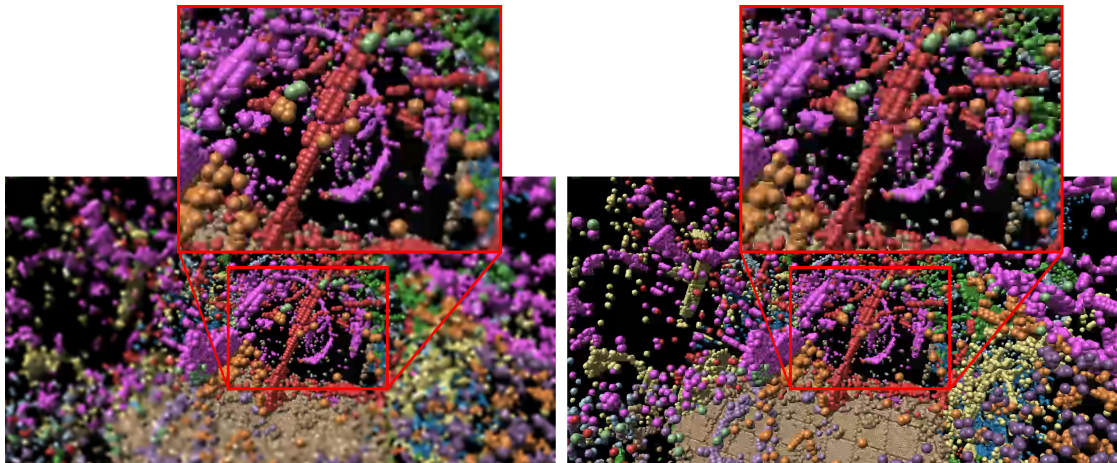
**Test Scenarios** As the M6000 has 12 Gb of VRAM, only one index and weight map pair can be used. Therefore, only two tests using the *crystal* dataset were performed. Both tests measured the throughput and all latencies while the user stood 0.5 metres and four metres away from the display. The results can be compared against the same tests carried out for the foveated encoding, which allows conclusions to be drawn regarding the required bandwidth when multiple maps are used.



**Figure 6.14** — The measured throughput for the two distances (0.5 and four metres) using the foveated encoding (■) and the stippling-based encoding (■). The foveated encoding used the contrast-based fall-off and both use QP remapping to reduce the QP value based on the distance. As can be seen, the stippling-based encoding reduces the throughput to approximately 70 Mb/s for both the near and the far distance compared to 85.3 Mb/s and 108.5 Mb/s for the foveated encoding.

**Test Results** As previously discussed for the foveated encoding (cf. subsection 6.2.3), the latency for the encoding changes only slightly compared to the normal encoding. The same is true for the stippling-based encoding, which produces similar latencies to the foveated encoding for most steps of the pipeline, see Figure 6.13, which contains the latencies for the *crystal* dataset. Both, the encoding and decoding latency, are reduced to the level of the low encoding setting. For the encoding latency the minimum and median is approximately 6 ms, while the maximum is around 8 ms. For the decoding, the minimum and median latency is around 5 ms, while the maximum is approximately 6 ms. The image reconstruction, which is included in the display queue latency, takes between 3.5 ms and 11 ms, while the median is about 5.2 ms. The latencies are similar for both distances, although slightly lower for the 0.5 metres test since there are fewer macroblocks inside the foveated region.

In addition to the latencies the throughput was measured. The test used the *crystal* dataset, while the user stood 0.5 meters and four meters away from the display and focused on a single point in the centre of the display. Figure 6.14 shows the measured throughput, in Mb/s, for the two tests as well as the foveated encoding using the contrast-based fall-off and the QP remapping. QP remapping was also used for the stippling-based encoding and it reduces the required bandwidth to the level of the normal encoding using the low setting. In the 0.5-metre test the maximum throughput was 68.8 Mb/s, compared to 63.5 Mb/s for the low setting. For the four-metre test the maximum throughput was only slightly higher with 71.3 Mb/s. The foveated encoding in comparison uses at most 85.3 Mb/s and 108.5 Mb/s for the same test. Overall, the stippling-based encoding reduces the throughput by at least 17 Mb/s, when the user is close to the display, and at most by 38 Mb/s, if the user is further away.



**Figure 6.15** — Image quality comparison between the stippling-based encoding (left) and the low encoding setting (right) using a single frame of the *crystal* dataset. As can be seen, the image quality close to the gaze point (red box) is higher when using the stippling-based encoding but the image quality in the peripheral region is lower. This is due to the fact that the peripheral region uses fewer samples and a high QP value, which both reduce the image quality.

Originally, the intention was to only use the colour values at the sample positions and leave the remaining frame empty. While this works relatively well close to the gaze point, due to the high number of samples, the results for the remaining samples in the periphery are inadequate. Either the encoder removes the colour values since they are the outlier inside a macroblock which contains predominately black pixels, or the encoder keeps the colour values which greatly increases the required bandwidth. The first occurs when using either the low or medium encoder settings, while the second occurs when using the high setting. To resolve this issue, multiple packing strategies were implemented ranging from packing all samples of a single row right next to each other, to finding an even distribution of samples in a smaller texture. The result was identical, either the decoded image lost colour intensity in the peripheral region or the throughput was increased. Since the encoding benefits from similar regions that change slowly over time the used packing strategy repeats the colour value of a sample within the Voronoi cell. This results in an image that has similar regions that can be encoded efficiently, although the resolution stays the same when compared to the foveated encoding. Overall the reduction in the throughput comes from the combination of the QP offset map and the repetition of colour values. However, this reduces the image quality in the peripheral region when compared with the foveated region since this region is sub-sampled and uses a high QP value. Despite this, the

perceived image quality is still much higher than with the normal encoding using the low setting, while requiring less bandwidth.

The end-to-end latency for both tests was hardly noticeable in side-to-side comparison, as it was at most 19 ms, and 13 ms on average. The major contribution to the overall latency, with up to 11 ms on the client side, comes from the image reconstruction. Similar to the foveated encoding, the latency for the other operations, especially encoding and decoding, ranges between the low setting, when no foveated region is present, and the high setting, if the region covers the whole display. During the tests the latter was at no time encountered and the encoding latency was on average 6 ms and at most 8 ms. Additionally, the latency showed no significant increase with multiple users.

The measured throughput for the stippling-based encoding at no time exceeded 72 Mb/s, therefore the stippling-based encoding requires on average between 5 and 8 Mb/s more than the low setting. If the QP remapping is not used the throughput increases slightly to approximately 80 Mb/s. Tests carried out with two users showed no significant increase in the measured throughput. All in all, results showed that stippling-based encoding, using QP remapping, uses, on average, 13 percent of the measured throughput required for the normal encoding using the medium encoder setting and 3 percent compared to the high encoder setting, while providing the same quality locally in the areas the users are focusing on, see Figure 6.15. However, the expected throughput reduction was higher due to the fact that fewer than 10% of the pixels of the captured frame are required for the reconstruction. The discrepancy between the expected and the measured reduction can mostly be attributed to the fact that it is difficult to use a video encoder for sparse frames with the goal of keeping the sparse data, i.e. the colour value of the pixels.

The size of the index and weight map limits the usage of this approach for GPU based hardware encoding, since multiple maps are required to cover all positions in front of the display. This is less of a problem when using CPU based encoding. While the encoding latency increases slightly due to the software based encoding, CPUs can provide more than enough RAM to load multiple maps. Furthermore, the stippling approach also works with other rendering techniques, such as raytracing or volume rendering, reducing the computation time significantly, especially since the computationally expensive part, i.e. the natural neighbour interpolation, can be moved to a different (remote) node.

## 6.4 Summary and Conclusion

Both approaches discussed in this chapter reduce the required bandwidth significantly for remote visualisation on large high-resolution displays. Additionally, both reduce the latency, especially the maximum decoding latency. The foveated encoding adapts

the quantisation value for each macroblock based on the users' gaze by either using foveated regions and a cubic visual acuity fall-off or a fall-off based on contrast sensitivity functions. The stippling-based encoding also adapts the quantisation value for each macroblock based on the users' gaze and additionally applies a sample map prior to encoding. This map contains sample positions of a 2D Gaussian function determined by a modified Linde-Buzo-Gray algorithm. Therefore, the frames that are encoded only contain the colour values of the captured frame at the sample positions reducing the amount of information that needs to be encoded. This reduces the required throughput further, but also decreases the image quality in the peripheral region due to the reduced sampling and the higher QP values. While both methods can be used to reduce the required bandwidth they provide a trade-off between quality in the peripheral region and the required bandwidth. If the image quality is more important the foveated encoding should be used, in combination with the QP remapping, as this combination already reduces the required bandwidth significantly. However, if this reduction is not enough and the image quality in the peripheral region is not as important the stippling-based encoding provides a similar image quality while requiring less bandwidth. The QP remapping should be used regardless of the method as it has no impact on the image quality while also reducing the required bandwidth, especially if the user is further away from the display.

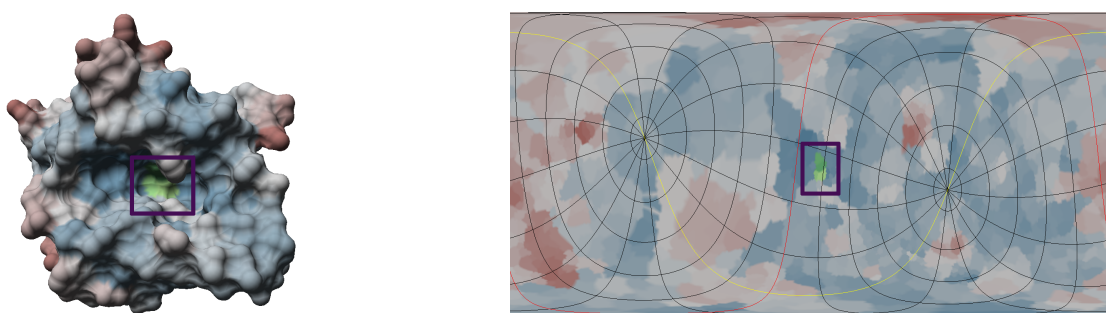
## DISCUSSION AND CONCLUSION

### Parts of this chapter have been published in:

- M. Krone, F. Frieß, K. Scharnowski, G. Reina, S. Fademrecht, T. Kulschewski, J. Pleiss, and T. Ertl. Molecular Surface Maps. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):701–710, 2017. [86]
- K. Schatz, F. Frieß, M. Schäfer, P. C. F. Buchholz, J. Pleiss, T. Ertl, and M. Krone. Analyzing the similarity of protein domains by clustering Molecular Surface Maps. *Computers & Graphics*, 99:114–127, 2021. [140]
- F. Frieß, M. Becher, G. Reina, and T. Ertl. Amortised Encoding for Large High-Resolution Displays. *IEEE 11th Symposium on Large Data Analysis and Visualization*, pages 53–62, 2021. [43]

This chapter evaluates the methods presented in the previous chapters regarding their suitability to solve actual research questions. As mentioned in the introduction, the most important concept for the methods developed for this thesis was the interactive and uninterrupted visual exploration of scientific visualisations in a remote scenario for large high-resolution displays. Therefore, the first part of the thesis presented methods that allow for visual exploration of molecular simulations, while the second part discussed how to extend these methods so that they can be used interactively in a remote scenario for large high-resolution displays. The two-dimensional representation of the molecular surface, discussed in chapter 3, that shows both the physico-chemical properties as well as the topography, allows for a visual exploration of molecular dynamic simulations. This helps analysts to better understand the data and to gain new

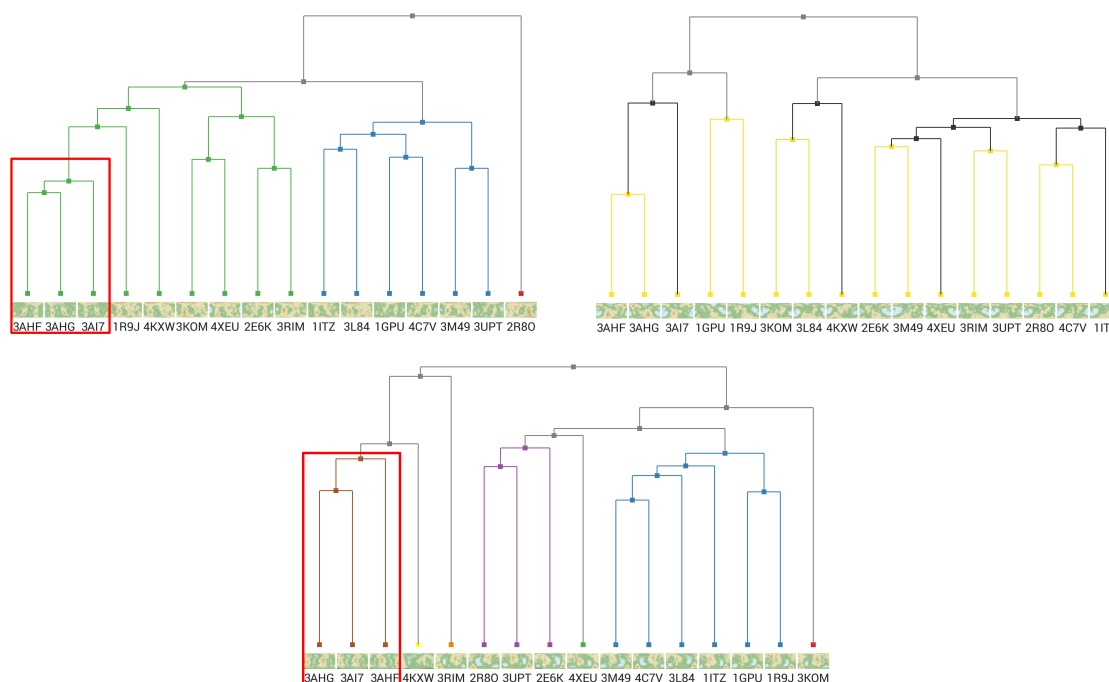




**Figure 7.1** — Analysis of a lipase data set (PDB ID: 1EX9) using Molecular Surface Maps. Two colouring modes are overlaid on the SES (left) and the map (right): colour according to the temperature factor (blue-red gradient) and colouring by binding site (green spot, surrounded by the purple box). As can be clearly observed on the map, the green binding site is located in a stable area of the protein (blue). The map was created using the parameter-based method and Hobo-Dyer projection. © IEEE 2017 [86].

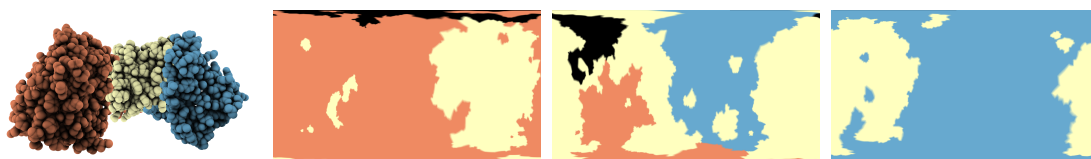
insights about the behaviour of the molecular systems under the simulated boundary conditions. The water binding of the CALB shown in Figure 3.10 is one example of how the maps can lead to new insights. Another example is the analysis of a lipase data set (PDB IDs: 1CRL, 1EX9, 1TCA, 1THG, 3TGL). Although lipases share a common fold, there are considerable differences between the various lipases. All lipases catalyse the same type of reactions, however, there are differences in their optimal temperature and their substrate preference. A comparison of both the hydrophobicity and the temperature factor of the aforementioned lipases showed that their surface properties differ largely from each other. While the protein surface of most lipases is divided into many small hydrophilic and hydrophobic patches, the surfaces of 1EX9 and 1CRL present larger homogeneous patches. Investigation of the temperature factor, which is an indicator for the stability of the protein, revealed that all lipases are relatively stable with the exception of 1EX9, which has many large flexible regions. The binding site colouring of the maps can reveal whether a lipase is in the open state (i.e. the binding site is accessible at the surface) or in the closed state (binding site inaccessible). Identifying and understanding such differences in the protein interface is crucial for understanding and predicting the effect of mutations. With the Molecular Surface Maps, users get an overview of these surface properties. The project partners especially liked having the possibility of overlaying two maps of the same protein, since this can also reveal interesting phenomena. This, for example, showed that although lipase 1EX9 is overall much more flexible than the other four lipases, the binding site is not only accessible but also lies in a very stable region (see Figure 7.1). Comparing maps of different proteins side by side allows users to assess their differences in a quick and convenient way compared to conventional molecular visualisation tools.





**Figure 7.2** — Clustering hierarchy of the PYR domain (top left) and the TKC domain (top right) and the PP domain (bottom) of the ketolase data set, clustered by height map and hydrophobicity. Top left: the three phosphoketolases are clustered together early (red box), while the two that are evolutionary the closest (3AHF and 3AHG) merge first. Top right: the leaf node merges of the dendrogram are coloured according to TM-score (using the Viridis colour map, see Figure 3.15). As observable, the TM-score exhibits a very strong agreement with the clustering also for this data set, with only negligible differences. Bottom: the clustering of the PP domains of the phosphoketolases differs slightly from the one for PYR and TKC domains; however, all three phosphoketolases are still in the same cluster (red box). © Elsevier 2021 [140].

The Molecular Surface Map can also be used to hierarchically cluster and analyse protein ensembles based on surface similarity, which implies a similar function. In order to construct a dendrogram that represents the clustering hierarchy feature vectors are extracted, using a CNN, from multiple maps that show different properties of the surface. The resulting clustering can be explored and analysed interactively. Additionally, a detailed analysis of individual pairs of proteins is possible in two 3D views, which helps to assess the quality of the map-based similarity score. As an example, the clustering was successfully applied to a detailed comparative analysis of the domains of proteins that belong to the same family and are, thus, structurally and functionally similar. Here, the visualisation was instrumental for the in-depth analysis of the different domains



**Figure 7.3** — Van der Waals surface and Molecular Surface Maps for the three domains in a monomer of human transketolase (PDB ID: 4KXW). Maps from left to right: Molecular Surface Map of the N-terminal PP domain, the central PYR domain, and the C-terminal TKC domain. The base colour of the maps are as follows: PP domain (■), PYR domain (■), and the TKC domain (■). Where a domain is in contact with another domain (distance  $< 5 \text{ \AA}$ ), the colour of the other domain is used instead. In addition, the contact to the cofactor thiamine diphosphate (or an analogous molecule, if available) is also depicted (■). In the surface view to the left, the cofactor is not visible as it is embedded between the PP and the PYR domain. © Elsevier 2021 [140].

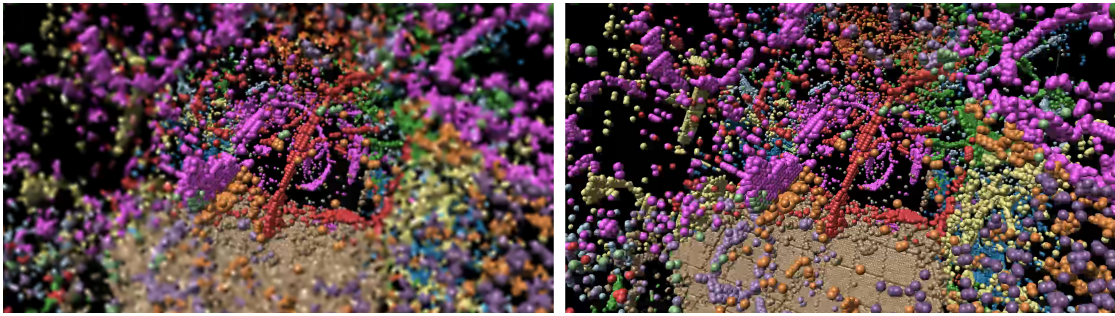
and their subtle differences.

The clustering was used to analyse an ensemble consisting of thirteen protein structures of transketolases and three protein structures of phosphoketolases that were selected previously for a representative multiple sequence alignment of both protein families [10]. Transketolases (TKs) are important enzymes in carbohydrate metabolism that catalyse the cleavage of carbon-carbon bonds in a ketose and the transfer of the remaining two-carbon unit onto an aldose [79]. Phosphoketolases (PKs) are enzymes similar to TKs that occur in a variation of the carbon metabolism that was found in certain fermentative bacteria such as *Bifidobacterium* sp. [132]. Both TKs and PKs require the cofactor molecule thiamine diphosphate (also called thiamine pyrophosphate) as a catalyst and share a common arrangement of protein domains, despite local differences in their structures [35, 165]. A TK or PK monomer includes an N-terminal PP domain, a central PYR domain and a C-terminal TKC domain. The PP domain and the PYR domain interact with the cofactor thiamine diphosphate and are thus required for the catalytic function, whereas the role of the TKC domain is not completely elucidated [30]. The domain structures of the phosphoketolases (PDB accessions 3AHF, 3AHG, and 3AI7) were found to cluster separately from the remaining thirteen transketolase representatives (cf. Figure 7.2), which was expected since protein structures from the phosphoketolase subfamily differ from transketolase structures, e.g. in loop regions of the PYR domain [165]. This behaviour can be observed for all of the three created trees. We observed slight deviations between the clusterings of the individual protein domains (see Figure 7.2), which can be explained by different degrees of conservation, with protein sequences of PYR domains usually being more conserved than protein sequences of PP domains, for instance [165]. As an example, in the case of the PP domains, PDB IDs 3AI7 and 3AHG were clustered in one subgroup,

i.e. as being more similar to each other than to the also quite similar PP domain of PDB ID 3AHF, whereas the PYR domains of these two PDB entries were split in two subgroups, as the PYR domains for PDB IDs 3AHF and 3AHG were clustered in one subgroup instead. All three investigated phosphoketolase structures originated from *Bifidobacterium* sp., whereas the selected transketolase representatives originated from more diverse taxa, ranging from bacteria to eukaryotes. The taxonomic hierarchy of the source organisms, however, does not necessarily imply an equivalent hierarchical clustering of the separate protein domains, as the functional and visual differences might not reflect this directly. Using the tile-based approach led to a more distinct clustering of the phosphoketolases. However, solely based on protein function, the domain scientists argued that a strong clustering of the phosphoketolases is essential and should indicate a positive result, which is provided by the tile-based approach.

To further inspect the relations between the different domains, the contact maps were used together with the highlighting of the viewed area (terminator). Closer inspection of the Molecular Surface Maps of the N-terminal PP domain showed interactions with the PYR domain, but not with the TKC domain (for an example, see Figure 7.3); the Molecular Surface Maps of the central PYR domain showed interactions with both the PP domain and the TKC domain; and the Molecular Surface Maps for the C-terminal TKC domain showed interactions with the PYR domain only. These observations are in agreement with the known domain arrangement of transketolases and phosphoketolases [35] and could be drawn by three superficial observations on the resulting map clusterings. Whereas the molecular surface maps of the TKC domain did not show any interaction with the cofactor thiamine diphosphate, the maps for both the PP and the PYR domain showed a distinct region of cofactor interaction, representing the approximate location of the active site. In contrast to other surface-based methods, such conclusions can be quickly drawn, as the users do not have to rotate the protein to perceive the complete surface. If the user is unsure about the exact location of the investigated region, the area highlighting can provide a visual aid.

The system discussed in chapter 4 lays the foundation for the interactive uninterrupted remote visual exploration of scientific visualisations for large high-resolution displays. Its architecture and implementation are designed in such a way that the system can run in different kinds of tiled display environments, while utilising any hardware support available, e.g. GPU based video encoder and decoder. To achieve this, a declarative approach via XML is used, which allows the configuration of each node in the system in the same file. This approach allows different scenarios to be run ranging from screen sharing to typical teleconference scenarios, or a combination of both, i.e. screen sharing with additional bidirectional 4K video stream as well as bidirectional audio streams. One example of how the system was used is the lecture series, which is an integral part



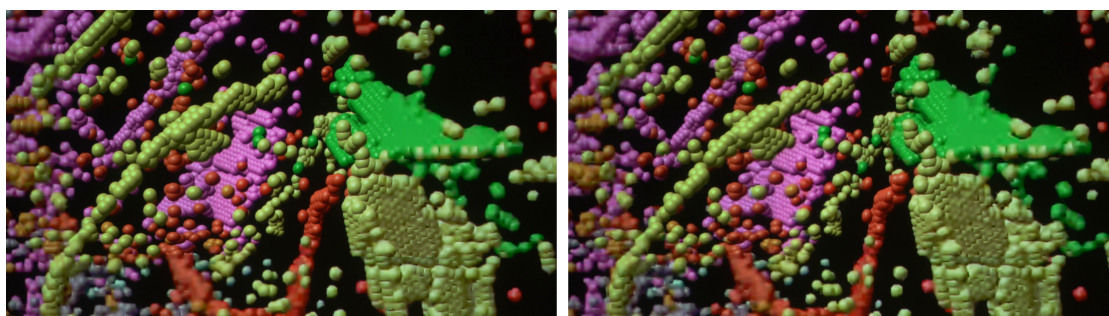
**Figure 7.4** — Image quality comparison between the stippling-based encoding (left) and the foveated encoding (right) using a single frame of the *crystal* dataset. As can be seen, close to the gaze point there is little difference between the two methods, while the foveated encoding produces a higher image quality in the peripheral region.

of the doctoral qualification program of SFB/Transregio 161<sup>1</sup>, where the system has been used on a weekly basis for the last three years.

Building upon the system, the optimisations discussed in chapter 5 and chapter 6 reduce the bandwidth and latency required to transmit the full resolution display, while keeping the high image quality. While the algorithmic optimisations, the adaptive encoding and the amortised encoding (see section 5.1 and section 5.2) work with the information contained in the image and downscaling and up-scaling, the user-driven optimisations, the foveated and the stippling-based encoding (see section 6.2 and section 6.3), work with eye tracking. All four reduce the required bandwidth but the user-driven approaches work best since they only increase the image quality in areas that are currently under investigation by the users. However, for both the foveated encoding and the stippling based encoding either eye tracking or an optical tracking system is required to support multiple users. For the adaptive and amortised encoding no such limitation exists. However, the adaptive encoding, as well as the stippling-based encoding and the foveated encoding, requires a GPU from NVIDIA since only the NVIDIA SDK provides access to the QP value of the macroblocks.

Although, the encoding and decoding latency of the baseline system is already low it can be reduced further by either using tiling or the amortised encoding. The foveated encoding as well as the stippling-based encoding also reduce the encoding and decoding latency slightly, while the adaptive encoding increases the server side latency due to the CNN predictions. However, this increase still results in interactive frame rates of about 25 fps. While the encoding and decoding latency do not change significantly

<sup>1</sup> SFB-TRR 161 “Quantitative Methods for Visual Computing”, <https://www.sfbtrr161.de/> (last accessed 22/04/2022)



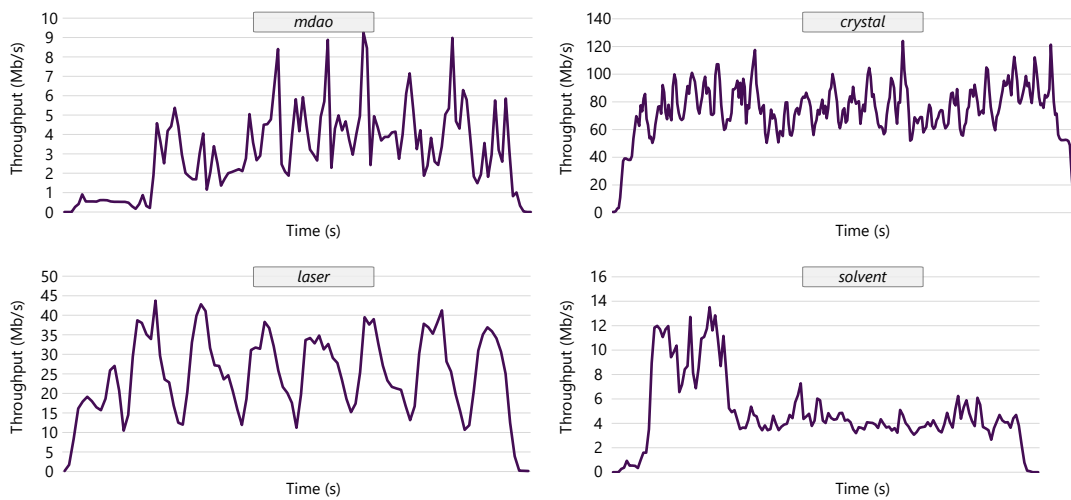
**Figure 7.5** — Two photos of the *crystal* dataset, that were encoded using the combination of amortised encoding and foveated encoding. The photo on the left shows a static scene resulting in a perfect reconstruction inside and outside of the foveated region, while the photo on the right shows a frame that is non-static, i.e. all coloured pixels are interpolated. As can be seen, the image quality of the photo on the right is slightly lower due to the fact that only one pattern was used to reconstruct it. However, the usage of the foveated encoding does not further negatively impact the perceived image quality.

depending on the load of the GPU, the time it takes to convert the frame does. For the *laser* dataset this increases the latency from at most 4 ms to around 30 ms, since the compute shader that performs the conversion has to wait for the rendering to finish. If the encoder supports the BRGA colour format this step can be skipped, otherwise it can be either moved to a different node, if there is enough bandwidth available to transmit the raw frame, or offloaded to a different GPU on the same node, if there is one available.

All optimisation methods reduce the required bandwidth by reducing the image quality without changing the perceived quality. The amortised encoding reduces the quality in non-static regions, the foveated and stippling-based encoding reduce the quality in the peripheral region and the adaptive encoding reduces the quality in regions with low details. Therefore, there is a visible difference in image quality between these methods. This can be seen in Figure 7.4, where the stippling-based encoding is shown on the left and the foveated encoding is shown on the right. The image quality close to the gaze point is similar for both, but the image quality in the peripheral region is better for the foveated encoding. However, this comes at the cost of an increase in throughput of approximately 30 Mb/s. This is a trade-off that can be made between higher overall image quality and lower throughput.

The methods discussed in chapter 5 and chapter 6 can be combined to further reduce the required bandwidth. For example, the amortised encoding (cf. section 5.2) and





**Figure 7.6** — Aggregated measured throughput of all nodes for the four datasets, *mdao* (top left), *crystal* (top right), *laser* (bottom left) and *solvent* (bottom right). The throughput measured for the combination of amortised and foveated encoding is similar to the foveated encoding for the *crystal* dataset and the *laser* dataset, while it requires less bandwidth for the *mdao* dataset as well as the *solvent* dataset.

foveated encoding (cf. section 6.2) can be combined. For this, the QP offset map is no longer computed for the full resolution frame but for each of the downsampled pattern frames. As mentioned before this only works with NVIDIAs SDK since the AMD and Intel SDKs do not provide functions to access and change the quantisation values of the macroblocks. A comparison between a non-static and a static frame that were both encoded using the combination can be seen in Figure 7.5. While the reconstruction lowers the image quality slightly, the additional foveated encoding does not further negatively impact the perceived image quality.

The encoding and decoding latency does not change significantly compared to the amortised encoding. This is in line with the foveated encoding (cf. subsection 6.2.3) for which both latencies correlate with the number of macroblocks inside the foveated region. Overall, the median encoding latency increases slightly from 3.4 ms to 3.6 ms while the maximum latency is unchanged at 6 ms. For the decoding latency a similar increase was measured. The maximum throughput for the combination of amortised and foveated encoding, using the *mdao* dataset, falls from 395 Mb/s to 9.3 Mb/s, see Figure 7.6. Using only the foveated encoding reduced the throughput to 15 Mb/s, down from 250 Mb/s for the normal encoding using the high setting. Although the amortised encoding alone increases the throughput, for this visualisation, compared to the normal full-resolution encoding using the high encoder setting, it decreases the throughput

for the low setting. This is the main contributing factor for the reduced bandwidth in this case as most of the image is encoded using the low setting. For the *solvent*, where the amortised encoding reduces the throughput regardless of the setting used, the combination uses at most 13.5 Mb/s, while the foveated encoding on its own uses at most 32 Mb/s.

In summary, the work conducted for this thesis focused on the interactive and uninterrupted remote visual exploration of scientific visualisations for large high-resolution displays. This included methods for rendering and clustering molecular surfaces as a 2D representation, solving the view-dependency and occlusion of the 3D representation. Particular attention was paid to making the representation easy to understand using the world map metaphor and by providing a variety of different map projections in order to help analysts gain new insights. Since surface similarity implies a similar function of the proteins, a hierarchically clustering of the 2D representations allows the analysis of protein ensembles. Furthermore, a system was developed that is built on pixel-streaming and allows the content of large-high resolution displays to be interactively shared, furthering the collaboration capabilities of such displays. Additionally, this system also offers teleconferencing scenarios, which can be coupled with the aforementioned screen sharing. Based on this system, further methods have been developed which focus on reducing the end-to-end latency as well as reducing the required bandwidth for the screen sharing while keeping the image quality as high as possible. Using low-quality encoding settings may greatly impact the user experience, e.g. resulting in the loss of fine details. Therefore, it is important for an interactive remote visualisation to offer the best image quality possible in order to assist domain scientists in understanding the results of their experiments, which can ultimately lead to new insights.

Although the techniques discussed in this thesis are tailored to biomolecular data and video encoding, several of them can be used in other domains as well. An example is the stippling-based encoding (see section 6.3). The technique behind this has already been used for volume rendering [23] but it could also be used for raytracing. For example, the CPU-based raytracing in MegaMol [121] can be adapted to use the same sample map used for the stippling-based encoding to significantly reduce the number of rays. Each pixel of the rendered image is then assigned the colour value of the closest sample, similar to the work done by the compute shader used for the stippling-based encoding (cf. subsection 6.3.2). The resulting frame can then be encoded, using the stippling-based encoding, skipping the Screen Capturing step, either on the GPU if one is available, or on the CPU, and transmitted to the client side. This would move the expensive part of the computation, i.e. the natural neighbour interpolation, to the client side, where it can be computed on a GPU. Additionally, this can either increase the frame rate or reduce the number of nodes used to render the frames when used for in-situ visualisation due to the significantly lower amount of rays. Furthermore, the lessons learned from making a sparse image better suited for video encoding so that it does not



introduce artefacts such as colour loss or colour bleeding, can be used for other data that is transmitted using encoders. An example of this type of data would be Smoothed Particle Hydrodynamics (SPH) for 3D-Sparse-Volumes.

The remote visualisation system and the optimisation methods developed for this thesis also provide a strong foundation for future research in this area. One opportunity for future research is to improve the collaborative part of the system by adding real-time collaborative text and code editing capabilities. This would allow the leverage of wall-sized displays for new usage scenarios and satisfy an actual need of researchers collaborating from different locations. The system could allow multiple users to interact with the same document, large parts of which are displayed on the large display, using a Laptop providing functions similar to Overleaf<sup>2</sup>. In addition to the text editing functionality, collaborative code analysis on large and high-resolution displays could be offered using syntax highlighting, word completion, go-to functions, auto formatting, re-naming, and other functionality. Additionally, assistive in-situ visualisation, e.g. overlays such as call graphs or dependency graphs, to help the user understand the code base, could be displayed, either directly on the display or using AR devices like the HoloLens.

Another promising direction of future work would be to further look into the minimal required QP value for a certain distance, with and without foveated encoding. As the pilot study (see subsection 6.2.3) showed, there is little difference between the QP values selected by the participants. This would help to potentially reduce the required bandwidth for all methods discussed in this thesis, since the pilot study indicates that a QP value of 22 is sufficient for a display with a resolution of  $10800 \times 4096$  and a distance of 50 cm for participants not to notice any difference to the original image. However, the pilot study only used the *crystal* dataset that only contains spheres. Therefore, multiple different visualisations that also contain finer structures need to be evaluated in order to find a minimal QP value based on distance. This will become increasingly important in the future as the resolution is certainly going to increase resulting in a need for additional bandwidth.

---

<sup>2</sup> <https://www.overleaf.com/> (last accessed 22/04/2022)





## BIBLIOGRAPHY

- [1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. *12th USENIX symposium on operating systems design and implementation*, pages 265–283, 2016. 51
- [2] S. J. Adelson and L. F. Hodges. Generating exact ray-traced animation frames by reprojection. *IEEE Computer Graphics and Applications*, 15(3):43–52, 1995. 105
- [3] M. Afonso, F. Zhang, and D. R. Bull. Video compression based on spatio-temporal resolution adaptation. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(1):275–280, 2019. 105
- [4] T. Akenine-Moller, E. Haines, and N. Hoffman. *Real-time rendering*. AK Peters/crc Press, 2019. 27
- [5] P. Almeida. *Proteins: concepts in biochemistry*. Garland Science, 2016. 30
- [6] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *Molecular Biology*, 215(3):403–410, 1990. 14
- [7] AMD. AMD Advanced Media Framework SDK. Online, last accessed 22/04/2022. <https://gpuopen.com/advanced-media-framework/>. 70, 120
- [8] S. Anzali, G. Barnickel, M. Krug, J. Sadowski, M. Wagener, J. Gasteiger, and J. Polanski. The comparison of geometric and electronic properties of molecular surfaces by neural networks: Application to the analysis of corticosteroid-binding globulin activity of steroids. *Computer-Aided Molecular Design*, 10(6):521–534, 1996. 30, 46
- [9] B. Bach, P. Dragicevic, D. Archambault, C. Hurter, and S. Carpendale. A Review of Temporal Data Visualizations Based on Space-Time Cube Operations. *EuroVis - STARS*, 1:23–41, 2014. 37
- [10] A. Baierl, A. Theorell, U. Mackfeld, P. Marquardt, F. Hoffmann, S. Moers, K. Nöh, P. C. F. Buchholz, J. Pleiss, and M. Pohl. Towards a mechanistic understanding of factors controlling the stereoselectivity of transketolase. *ChemCatChem*, 10(12):2601–2611, 2018. 154

- [11] M. S. Banks, A. B. Sekuler, and S. J. Anderson. Peripheral spatial vision: Limits imposed by optics, photoreceptors, and receptor pooling. *Journal of the Optical Society of America A*, 8(11):1775–1787, 1991. 128
- [12] S. Beck, A. Kunert, A. Kulik, and B. Froehlich. Immersive group-to-group telepresence. *IEEE Transactions on Visualization and Computer Graphics*, 19(4):616–625, 2013. 64
- [13] J. M. Berg, J. L. Tymoczko, L. Stryer, and N. D. Clarke. *Biochemistry*. W. H. Freeman, New York, NY, 5. ed., 4. print edition, 2002. 14
- [14] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, 2000. <http://www.pdb.org>. 25, 30, 43, 49, 53
- [15] T. Biedert, P. Messmer, T. Fogal, and C. Garth. Hardware-accelerated multi-tile streaming for realtime remote visualization. *Eurographics Symposium on Parallel Graphics and Visualization*, pages 33–43, 2018. 3, 65, 73
- [16] J. F. Blinn. A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982. 32
- [17] Blue Brain Project. Tide: Tiled Interactive Display Environment. Online, last accessed 22/04/2022. <https://github.com/BlueBrain/Tide>. 64
- [18] M. E. Bock, C. Garutti, and C. Guerra. Discovery of similar regions on protein surfaces. *Computational Biology*, 14(3):285–299, 2007. 46
- [19] M. R. Bolin and G. W. Meyer. A frequency based ray tracer. *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 409–418, 1995. 91
- [20] S. Bosse, D. Maniry, T. Wiegand, and W. Samek. A deep neural network for image quality assessment. *IEEE International Conference on Image Processing*, pages 3773–3777, 2016. 92
- [21] S. Bremm, T. von Landesberger, M. Heß, T. Schreck, P. Weil, and K. Hamacherk. Interactive visual comparison of multiple trees. *IEEE Conference on Visual Analytics Science and Technology*, pages 31–40, 2011. 47
- [22] A. Bringmann, S. Syrbe, K. Görner, J. Kacza, M. Francke, P. Wiedemann, and A. Reichenbach. The primate fovea: Structure, function and development. *Progress in Retinal and Eye Research*, 2018. 128, 142

- [23] V. Bruder, C. Schulz, R. Bauer, S. Frey, D. Weiskopf, and T. Ertl. Voronoi-based foveated volume rendering. *EuroVis (Short Papers)*, pages 67–71, 2019. 11, 13, 139, 143, 159
- [24] N. J. Burgoyne and R. M. Jackson. Predicting Protein Function from Surface Properties. In D. J. Rigden, editor, *From Protein Structure to Function with Bioinformatics*, pages 167–186. Springer Netherlands, 2009. 2, 30
- [25] D. Cai, X. He, Z. Li, W.-Y. Ma, and J.-R. Wen. Hierarchical clustering of www image search results using visual, textual and link information. *ACM International Conference on Multimedia*, pages 952–959, 2004. 47
- [26] T. Can, C.-I. Chen, and Y.-F. Wang. Efficient molecular surface generation using level-set methods. *Molecular Graphics and Modelling*, 25(4):442–454, 2006. 18
- [27] K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, and C.-L. Lei. Measuring the latency of cloud gaming systems. *Proceedings of ACM International Conference on Multimedia*, pages 1269–1272, 2011. 62
- [28] Z. Chen and C. Guillemot. Perceptually-friendly h.264/avc video coding based on foveated just-noticeable-distortion model. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 806–819, 2010. 126
- [29] M. L. Connolly. Analytical Molecular Surface Calculation. *Journal of Applied Crystallography*, 16(5):548–558, 1983. 17
- [30] S. J. Costelloe, J. M. Ward, and P. A. Dalby. Evolutionary analysis of the TPP-dependent enzyme family. *Molecular Evolution*, 66(1):36–49, 2007. 154
- [31] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, 1989. 80
- [32] O. Deussen, M. Spicker, and Q. Zheng. Weighted linde-buzo-gray stippling. *ACM Transactions on Graphics*, 36(6):1–12, 2017. 139, 140
- [33] T. K. Dey, F. Fan, and Y. Wang. An Efficient Computation of Handle and Tunnel Loops via Reeb Graphs. *ACM Transactions on Graphics*, 32(4):32:1–32:10, 2013. 39, 44
- [34] K. U. Doerr and F. Kuester. CGLX: A scalable, high-performance visualization framework for networked display environments. *IEEE Transactions on Visualization and Computer Graphics*, 17(3):320–332, 2011. 12, 65
- [35] R. G. Duggleby. Domain relationships in thiamine diphosphate-dependent enzymes. *Accounts of Chemical Research*, 39(8):550–557, 2006. 154, 155

- [36] H. Edelsbrunner. Deformable Smooth Surface Design. *Discrete & Computational Geometry*, 21(1):87–115, 1999. 32
- [37] S. Eilemann, D. Steiner, and R. Pajarola. Equalizer 2.0—Convergence of a Parallel Rendering Framework. *IEEE Transactions on Visualization and Computer Graphics*, 26(2):1292–1307, 2020. 12, 65
- [38] I. Epic Games. High-Quality Temporal Supersampling. Online, last accessed 22/04/2022. [http://advances.realtimerendering.com/s2014/#\\_HIGH-QUALITY\\_TEMPORAL\\_SUPERSAMPLING](http://advances.realtimerendering.com/s2014/#_HIGH-QUALITY_TEMPORAL_SUPERSAMPLING). 105
- [39] A. Febretti, A. Nishimoto, V. Mateevitsi, L. Renambot, A. Johnson, and J. Leigh. Omegalib: A multi-view application framework for hybrid reality display environments. *IEEE Virtual Reality*, pages 9–14, 2014. 12, 65
- [40] W. M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155(3760):279–284, 1967. 14, 47
- [41] J. Flusser. On the independence of rotation moment invariants. *Pattern Recognition*, 33(9):1405–1410, 2000. 48, 50
- [42] S. Frey, F. Sadlo, and T. Ertl. Balanced sampling and compression for remote visualization. *SIGGRAPH Asia Visualization in High Performance Computing*, page 1, 2015. 92
- [43] F. Frieß, M. Becher, G. Reina, and T. Ertl. Amortised Encoding for Large High-Resolution Displays. *IEEE 11th Symposium on Large Data Analysis and Visualization*, pages 53–62, 2021. 5, 61, 80, 91, 106, 107, 109, 110, 111, 112, 113, 117, 119, 120, 151
- [44] F. Frieß, M. Braun, V. Bruder, S. Frey, G. Reina, and T. Ertl. Foveated Encoding for Large High-Resolution Displays. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–10, 2020. 6, 61, 79, 123, 127, 129, 130, 139
- [45] F. Frieß, M. Landwehr, V. Bruder, S. Frey, and T. Ertl. Adaptive Encoder Settings for Interactive Remote Visualisation on High-Resolution Displays. *Proceedings of the IEEE Symposium on Large Data Analysis and Visualization - Short Papers*, pages 87–91, 2018. 5, 91, 96, 97, 102, 103, 104
- [46] F. Frieß, C. Müller, and T. Ertl. Real-Time High-Resolution Visualisation. *Proceedings of the Eurographics Symposium on Vision, Modeling, and Visualization*, pages 127–135, 2020. 5, 61, 63, 64, 66, 67



- [47] H. Fukuda and K. Tomii. Deepeca: an end-to-end learning framework for protein contact prediction from a multiple sequence alignment. *BMC Bioinformatics*, 21(1):10, 2020. 57
- [48] A. Gorodilov, D. Gavrilov, and D. Schelkunov. Neural networks for image and video compression. *International Conference on Artificial Intelligence Applications and Innovations*, pages 37–41, 2018. 105
- [49] J. Görtler, M. Spicker, C. Schulz, D. Weiskopf, and O. Deussen. Stippling of 2d scalar fields. *IEEE Transactions on Visualization and Computer Graphics*, 25(6):2193–2204, 2019. 139, 140
- [50] P. Gralka, M. Becher, M. Braun, F. Frieß, C. Müller, T. Rau, K. Schatz, C. Schulz, M. Krone, G. Reina, and T. Ertl. Megamol – a comprehensive prototyping framework for visualizations. *The European Physical Journal (Special Topics)*, 227: Particle Methods in Natural Science and Engineering(14):1817–1829, 2019. 5, 26
- [51] J. G. Greener, S. M. Kandathil, and D. T. Jones. Deep learning extends de novo protein modelling coverage of genomes using iteratively predicted structural constraints. *Nature communications*, 10(1):3977, 2019. 57
- [52] GROMACS development team. Gromacs xtc. Online, <https://manual.gromacs.org/documentation/current/reference-manual/file-formats.html#xtc>, last accessed 22/04/2022, 2021. 25
- [53] S. Grottel. *Point-based visualization of molecular dynamics data sets*. PhD thesis, University of Stuttgart, 2012. 5
- [54] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder. Foveated 3D graphics. *ACM Transactions on Graphics*, page 164, 2012. 13
- [55] T. Hägerstrand. What about people in Regional Science? *Papers in Regional Science*, 24(1):6–21, 1970. 37
- [56] S. Hargreaves and M. Harris. Deferred shading. *Game Developers Conference*, 2:31, 2004. 27
- [57] J. W. Harris and H. Stöcker. *Handbook of mathematics and computational science*. Springer Science & Business Media, 1998. 125
- [58] K. Hasegawa and K. Funatsu. New description of protein-ligand interactions using a spherical self-organizing map. *Bioorganic & Medicinal Chemistry*, 20(18):5410–5415, 2012. 30

- [59] J. Hass and P. Koehl. How round is a protein? Exploring protein structures for globularity using conformal mapping. *Mathematics of Biomolecules*, 1:26, 2014. 31, 33
- [60] P. Hermosilla, M. Krone, V. Guallar, P.-P. Vázquez, À. Vinacua, and T. Ropinski. Interactive gpu-based generation of solvent-excluded surfaces. *The Visual Computer*, 33(6):869–881, 2017. 18
- [61] R. Herzog, S. Kinuwaki, K. Myszkowski, and H.-P. Seidel. Render2mpeg: A perception-based framework towards integrating rendering and video compression. *Computer Graphics Forum*, 27(2):183–192, 2008. 92
- [62] C. Hofbauer, H. Lohninger, and A. Aszódi. SURFCOMP: A novel graph-based approach to molecular surface comparison. *Chemical Information and Computer Scientists*, 44(3):837–847, 2004. 46
- [63] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 92
- [64] M.-K. Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, 8(2):179–187, 1962. 48, 50
- [65] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Transactions on Graphics*, 21(3):693–702, 2002. 12
- [66] J. D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering*, 9(3):90–95, 2007. 32
- [67] H.-W. Hunziker. *The eye of the reader: Foveal and peripheral perception from letter recognition to the joy of reading*. Transmedia Zurich, 2006. 13
- [68] D. H. Huson, D. C. Richter, C. Rausch, T. DeZulian, M. Franz, and R. Rupp. Dendroscope: An interactive viewer for large phylogenetic trees. *BMC Bioinformatics*, 8(1):460, 2007. 47
- [69] G. Illahi, M. Siekkinen, and E. Masala. Foveated video streaming for cloud gaming. *IEEE 19th International Workshop on Multimedia Signal Processing*, pages 1–6, 2017. 126
- [70] G. K. Illahi, T. V. Gemert, M. Siekkinen, E. Masala, A. Oulasvirta, and A. Ylä-Jääski. Cloud Gaming with Foveated Video Encoding. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 16(1):1–24, 2020. 126

- [71] Intel. Intel oneAPI Video Processing Library. Online, last accessed 22/04/2022. <https://www.intel.com/content/www/us/en/developer/tools/oneapi/onevpl.html>. 70, 120
- [72] Internet Engineering Task Force. Opus Interactive Audio Codec. Online, last accessed 22/04/2022, 2011. <https://opus-codec.org/>. 68, 73
- [73] M. R. Jakobsen and K. Hornbæk. Up close and personal: Collaborative work on a high-resolution multitouch wall display. *ACM Transactions on Computer-Human Interaction*, 21(2):11:1–11:34, 2014. 3
- [74] G. P. Johnson, G. D. Abram, B. Westing, P. Navrátil, and K. Gaither. DisplayCluster: An interactive visualization environment for tiled displays. *Proceedings of IEEE International Conference on Cluster Computing*, pages 239–247, 2012. 64
- [75] W. Kabsch and C. Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–2637, 1983. 49
- [76] L. Kang, P. Ye, Y. Li, and D. Doermann. Convolutional neural networks for no-reference image quality assessment. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1733–1740, 2014. 92
- [77] M. Klapperstueck, T. Czauderna, C. Goncu, J. Glowacki, T. Dwyer, F. Schreiber, and K. Marriott. Contextuwall: Multi-site collaboration using display walls. *Visual Languages and Computing*, 46:35–42, 2018. 63
- [78] M. A. Koch and H. Waldmann. Protein structure similarity clustering and natural product structure as guiding principles in drug discovery. *Drug Discovery Today*, 10(7):471–483, 2005. 15
- [79] G. A. Kochetov and O. N. Solovjeva. Structure and functioning mechanism of transketolase. *Biochimica et Biophysica Acta - Proteins and Proteomics*, 1844(9):1608–1618, 2014. 154
- [80] P. Koehl. Protein structure similarities. *Current Opinion in Structural Biology*, 11(3):348–353, 2001. 15
- [81] I. Kolesár, J. Byška, J. Parulek, H. Hauser, and B. Kozlíková. Unfolding and interactive exploration of protein tunnels and their dynamics. *Eurographics Workshop on Visual Computing for Biology and Medicine*, pages 1–10, 2016. 55
- [82] D. Koller, M. Turitzin, M. Levoy, M. Tarini, G. Croccia, P. Cignoni, and R. Scopigno. Protected interactive 3d graphics via remote rendering. *ACM Transactions on Graphics*, 23(3):695–703, 2004. 92

- [83] D. G. Kontopoulos, D. Vlachakis, G. Tsiliki, and S. Kossida. Structuprint: a scalable and extensible tool for two-dimensional representation of protein surfaces. *BMC Structural Biology*, 16(1):1–8, 2016. 31
- [84] B. Kozlíková, M. Krone, M. Falk, N. Lindow, M. Baaden, D. Baum, I. Viola, J. Parulek, and H.-C. Hege. Visualization of biomolecular structures: State of the art revisited. *Computer Graphics Forum*, 36(8):178–204, 2017. 15, 18, 29
- [85] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 92
- [86] M. Krone, F. Frieß, K. Scharnowski, G. Reina, S. Fademrecht, T. Kulschewski, J. Pleiss, and T. Ertl. Molecular Surface Maps. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):701–710, 2017. 5, 29, 33, 34, 35, 36, 39, 42, 45, 46, 48, 151, 152
- [87] M. Krone, S. Grottel, and T. Ertl. Parallel Contour-Buildup Algorithm for the Molecular Surface. *IEEE Symposium on Biological Data Visualization*, pages 17–22, 2011. 18
- [88] M. Krone, D. Kauker, G. Reina, and T. Ertl. Visual Analysis of Dynamic Protein Cavities and Binding Sites. *IEEE Pacific Visualization Symposium*, 1:301–305, 2014. 38
- [89] M. Krone, G. Reina, C. Schulz, T. Kulschewski, J. Pleiss, and T. Ertl. Interactive Extraction and Tracking of Biomolecular Surface Features. *Computer Graphics Forum*, 32(3):331–340, 2013. 38, 44
- [90] J. Krüger and R. Westermann. Acceleration Techniques for GPU-based Volume Rendering. *IEEE Visualization*, pages 287–292, 2003. 37
- [91] D. La, J. Esquivel-Rodríguez, V. Venkatraman, B. Li, L. Sael, S. Ueng, S. Ahrendt, and D. Kihara. 3d-surfer: software for high-throughput protein surface comparison and analysis. *BMC Bioinformatics*, 25(21):2843–2844, 2009. 37, 46
- [92] B. Lee and F. M. Richards. The interpretation of protein structures: estimation of static accessibility. *Journal of Molecular Biology*, 55(3):379–400, 1971. 17
- [93] M. Levoy. Volume rendering by adaptive refinement. *The Visual Computer*, 6(1):2–7, 1990. 91

- [94] C. Li, A. C. Bovik, and X. Wu. Blind image quality assessment using a general regression neural network. *IEEE Transactions on Neural Networks*, 22(5):793–799, 2011. 92
- [95] N. Lindow, D. Baum, and H.-C. Hege. Voronoi-based extraction and visualization of molecular paths. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2025–2034, 2011. 39, 44
- [96] R. Lipinski, K. Moreland, M. E. Papka, and T. Marrinan. Gpu-based Image Compression for Efficient Compositing in Distributed Rendering Applications. *IEEE 11th Symposium on Large Data Analysis and Visualization*, pages 43–52, 2021. 65
- [97] E. Liu. DLSS 2.0 - Image reconstruction for real-time rendering with deep learning. Online, last accessed 22/04/2022. <http://behindthepixels.io/assets/files/DLSS2.0.pdf>. 106
- [98] Z. Liu. *Lacome: a cross-platform multi-user collaboration system for a shared large display*. PhD thesis, University of British Columbia, 2007. 64
- [99] I. Live Networks. live555. Online, last accessed 22/04/2022. [www.live555.com](http://www.live555.com). 70
- [100] A. Löffler, L. Pica, H. Hoffmann, and P. Slusallek. Synchronous networked displays for vr applications: Display as a service (DaaS). *Proceedings of EuroVR*, 2012. 62
- [101] R. MacKenzie, K. Hawkey, K. S. Booth, Z. Liu, P. Perswain, and S. S. Dhillon. Lacome: A multi-user collaboration system for shared large displays. *Proceedings of ACM Conference on Computer Supported Cooperative Work Companion*, pages 267–268, 2012. 64
- [102] M. Maheshwari, S. Silakari, and M. Motwani. Image clustering using color and texture. *First International Conference on Computational Intelligence, Communication Systems and Networks*, pages 403–408, 2009. 48, 51
- [103] T. Marrinan, J. Aurisano, A. Nishimoto, K. Bharadwaj, V. Mateevitsi, L. Renambot, L. Long, A. Johnson, and J. Leigh. Sage2: A new approach for data intensive collaboration using scalable resolution shared displays. *Proceedings of IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 177–186, 2014. 3, 61, 63
- [104] T. Marrinan, S. Rizzi, J. A. Insley, L. Long, L. Renambot, and M. E. Papka. Pxstream: Remote visualization for distributed rendering frameworks. *IEEE 9th Symposium on Large Data Analysis and Visualization*, pages 37–41, 2019. 3, 65

## 172 Bibliography

- [105] Microsoft. Desktop Duplication API. Online, last accessed 22/04/2022, 2018. <https://docs.microsoft.com/en-us/windows/desktop/direct3ddxgi/desktop-dup-api>. 68, 95, 109, 144
- [106] Microsoft. I/O Completion Ports. Online, last accessed 22/04/2022, 2018. <https://docs.microsoft.com/en-us/windows/desktop/fileio/i-o-completion-ports>. 76
- [107] U. Montreal. Rendering 'Rainbow Six | Siege'. Online, last accessed 22/04/2022. <https://www.gdcvault.com/play/1022990/Rendering-Rainbow-Six-Siege>. 105
- [108] K. Moreland, D. Lepage, D. Koller, and G. Humphreys. Remote rendering for ultrascale data. *Journal of Physics: Conference Series*, 125(1):012096, 2008. 92
- [109] J. Moulton, J. T. Pedersen, R. Judson, and K. Fidelis. A large-scale experiment to assess protein structure prediction methods. *Proteins: Structure, Function, and Bioinformatics*, 23(3):ii–iv, 1995. 14
- [110] NVIDIA. NVIDIA Quadro Sync. Online, last accessed 22/04/2022. <https://www.nvidia.com/en-us/design-visualization/solutions/quadro-sync>. 77, 114
- [111] NVIDIA. NVIDIA Video Codec SDK. Online, last accessed 22/04/2022. <https://developer.nvidia.com/nvidia-video-codec-sdk>. 70, 120
- [112] Omar Cornut. ImGui Project Page. Online, <https://github.com/ocornut/imgui>, last accessed 22/04/2022, 2014. 27
- [113] D. Pajak, R. Herzog, E. Eisemann, K. Myszkowski, and H.-P. Seidel. Scalable remote rendering with depth and motion-flow augmented streaming. *Computer Graphics Forum*, 30(2):415–424, 2011. 92
- [114] S. Pandey and P. Khanna. A hierarchical clustering approach for image datasets. *International Conference on Industrial and Information Systems*, pages 1–6, 2014. 47
- [115] A. A. Polyansky, A. O. Chugunov, P. E. Volynsky, N. A. Krylov, D. E. Nolde, and R. G. Efremov. Preddimer: a web server for prediction of transmembrane helical dimers. *BMC Bioinformatics*, 30(6):889–890, 2014. 31
- [116] K. Ponto, K. Doerr, T. Wypych, J. Kooker, and F. Kuester. CGLXTouch: A multi-user multi-touch approach for ultra-high-resolution collaborative workspaces. *Future Generation Computer Systems*, 27(6):649–656, 2011. 65

- [117] N. Postarnakevich and R. Singh. Global-to-local representation and visualization of molecular surfaces using deformable models. *ACM Symposium on Applied Computing*, pages 782–787, 2009. 30, 33
- [118] S. J. Rahi and K. Sharp. Mapping complicated surfaces onto a sphere. *International Journal of Computational Geometry & Applications*, 17(04):305–329, 2007. 31, 34, 40, 41
- [119] G. Ramachandran, C. Ramakrishnan, and V. Sasisekharan. Stereochemistry of polypeptide chain configurations. *Journal of Molecular Biology*, 7(1):95–99, 1963. 30
- [120] A. Rao and R. Lanphier. Real Time Streaming Protocol(RTSP). Internet-Draft draft-rao-rtsp-00, Internet Engineering Task Force, 1996. Work in Progress. 19
- [121] T. Rau, M. Krone, G. Reina, and T. Ertl. Challenges and opportunities using software-defined visualization in megamol. *7th Workshop on Visual Analytics, Information Visualization and Scientific Visualization*, 2017. 159
- [122] T. Rau, S. Zahn, M. Krone, G. Reina, and T. Ertl. Interactive cpu-based ray tracing of solvent excluded surfaces. *Eurographics Workshop on Visual Computing for Biology and Medicine*, 2019. 18
- [123] L. Renambot, T. Marrinan, J. Aurisano, A. Nishimoto, V. Mateevitsi, K. Bharadwaj, L. Long, A. Johnson, M. Brown, and J. Leigh. SAGE2: A collaboration portal for scalable resolution displays. *Future Generation Computer Systems*, 54:296–305, 2016. 3, 61, 63
- [124] L. Renambot, A. Rao, R. Singh, B. Jeong, N. Krishnaprasad, V. Vishwanath, V. Chandrasekhar, N. Schwarz, A. Spale, C. Zhang, G. Goldman, J. Leigh, and A. Johnson. SAGE: the Scalable Adaptive Graphics Environment. *Proceedings of WACE*, pages 2004–2009, 2004. 3, 63
- [125] F. M. Richards. Areas, Volumes, Packing, and Protein Structure. *Annual Review of Biophysics and Bioengineering*, 6(1):151–176, 1977. 17, 25
- [126] I. E. Richardson. *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*. John Wiley & Sons, 2004. 19, 22, 25
- [127] M. Rittenbruch. CubIT: Large-scale Multi-user Presentation and Collaboration. *Proceedings of International Conference on Interactive Tabletops and Surfaces*, pages 441–444, 2013. 63



- [128] J. Robson and N. Graham. Probability summation and regional variation in contrast sensitivity across the visual field. *Vision Research*, 21(3):409–418, 1981. 128
- [129] L. Sael, D. La, B. Li, R. Rustamov, and D. Kihara. Rapid comparison of properties on protein surface. *Proteins: Structure, Function, and Bioinformatics*, 73(1):1–10, 2008. 46
- [130] L. Sael, B. Li, D. La, Y. Fang, K. Ramani, R. Rustamov, and D. Kihara. Fast protein tertiary structure retrieval based on global surface shape similarity. *Proteins: Structure, Function, and Bioinformatics*, 72(4):1259–1273, 2008. 46
- [131] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987. 47
- [132] B. Sánchez, M. Zúñiga, F. González-Candelas, C. G. de los Reyes-Gavilán, and A. Margolles. Bacterial and eukaryotic phosphoketolases: phylogeny, distribution and evolution. *Molecular Microbiology and Biotechnology*, 18(1):37–51, 2010. 154
- [133] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted residuals and linear bottlenecks. *Proceedings of the IEEE conference on computer vision and pattern Recognition*, pages 4510–4520, 2018. 48, 51
- [134] Sanghoon Lee and A. C. Bovik. Fast algorithms for foveated video processing. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 149–162, 2003. 126
- [135] M. Sanner. *Sur la modélisation des surfaces moléculaires*. PhD thesis, Mulhouse, 1992. 18
- [136] M. F. Sanner, A. J. Olson, and J.-C. Spehner. Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers*, 38(3):305–320, 1996. 18, 43
- [137] A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C.-T. Lin. A review of clustering techniques and developments. *Neurocomputing*, 267:664–681, 2017. 46
- [138] K. Scharnowski, M. Krone, G. Reina, T. Kulschewski, J. Pleiss, and T. Ertl. Comparative visualization of molecular surfaces using deformable models. *Computer Graphics Forum*, 33(3):191–200, 2014. 30, 34, 40, 46
- [139] K. Schatz, J. J. Franco-Moreno, M. Schäfer, A. S. Rose, V. Ferrario, J. Pleiss, P.-P. Vázquez, T. Ertl, and M. Krone. Visual analysis of large-scale protein-ligand interaction data. *Computer Graphics Forum*, 40(6):394–408, 2021. 36

- [140] K. Schatz, F. Frieß, M. Schäfer, P. C. F. Buchholz, J. Pleiss, T. Ertl, and M. Krone. Analyzing Protein Similarity by Clustering Molecular Surface Maps. *Computers & Graphics*, 99:114–127, 2021. 5, 14, 29, 49, 57, 58, 151, 153, 154
- [141] K. Schatz, F. Frieß, M. Schäfer, T. Ertl, and M. Krone. Analyzing Protein Similarity by Clustering Molecular Surface Maps. *Eurographics Workshop on Visual Computing for Biology and Medicine*, pages 103–114, 2020. 5, 48, 52, 55, 56
- [142] K. Schatz, M. Krone, T. L. Bauer, V. Ferrario, J. Pleiss, and T. Ertl. Molecular Sombreros: Abstract Visualization of Binding Sites within Proteins. 2019. 31
- [143] C. Schulz, K. C. Kwan, M. Becher, D. Baumgartner, G. Reina, O. Deussen, and D. Weiskopf. Multi-class inverted stippling. *ACM Transactions on Graphics*, 40(6), 2021. 143
- [144] H. Schulzrinne, A. Rao, and R. Lanphieer. Rtpsp'. 1996. 19
- [145] M. Segal and K. Akeley. *The OpenGL Graphics System: A Specification (Version 4.6 (Core Profile))*. Khronos Group Inc., 2019. 9
- [146] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui. Cloud gaming: architecture and performance. *IEEE Network*, 27(4):16–21, 2013. 62
- [147] H. R. Sheikh, B. L. Evans, and A. C. Bovik. Real-time foveation techniques for low bit rate video coding. *Real-Time Imaging*, 9(1):27–40, 2003. 132
- [148] T. Shen, X. Huang, H. Li, E. Kim, S. Zhang, and J. Huang. A 3d laplacian-driven parametric deformable model. *International Conference on Computer Vision*, pages 279–286, 2011. 41
- [149] J. P. Snyder and P. M. Voxland. *An album of map projections*. Number 1453. US Government Printing Office, 1989. 35
- [150] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 2(38):1409–1438, 1958. 47, 48, 51, 52
- [151] Steinberg. Audio Stream Input/Output (ASIO). Online, last accessed 22/04/2022. <https://www.steinberg.net/en/company/technologies.html>. 68, 71
- [152] H. Steinhaus. Sur la division des corp matériels en parties. *Bulletin L'Académie Polonaise des Science*, 1(804):801, 1956. 46
- [153] M. Stengel, S. Grogorick, M. Eisemann, and M. Magnor. Adaptive image-space sampling for gaze-contingent real-time rendering. *Computer Graphics Forum*, pages 129–139, 2016. 13

- [154] H. Strasburger, I. Rentschler, and M. Jüttner. Peripheral vision and pattern recognition: A review. *Journal of Vision*, pages 13–13, 2011. 4, 13, 128, 142
- [155] K. Takahashi, T. Naemura, and M. Tanaka. Rate-distortion analysis of super-resolution image/video decoding. *18th IEEE International Conference on Image Processing*, pages 1629–1632, 2011. 105
- [156] D. Terzopoulos, A. Witkin, and M. Kass. Constraints on deformable models: Recovering 3d shape and nonrigid motion. *Artificial intelligence*, 36(1):91–123, 1988. 34
- [157] B. Tian. *Bouncing Bubble: A fast algorithm for Minimal Enclosing Ball problem*. GRIN Publishing, 2012. 40
- [158] M. Totrov and R. Abagyan. The contour-buildup algorithm to calculate the analytical molecular surface. *Structural Biology*, 116(1):138–143, 1996. 18
- [159] Y. Y. Tseng and W.-H. Li. Classification of protein functional surfaces using structural characteristics. *Proceedings of the National Academy of Sciences*, 109(4):1170–1175, 2012. 2
- [160] Y. Y. Tseng and W.-H. Li. Classification of protein functional surfaces using structural characteristics. *Proceedings of the National Academy of Sciences*, 109(4):1170–1175, 2012. 30
- [161] K. Vaidyanathan, M. Salvi, R. Toth, T. Foley, T. Akenine-Möller, J. Nilsson, J. Munkberg, J. Hasselgren, M. Sugihara, P. Clarberg, et al. Coarse pixel shading. *Proceedings of High Performance Graphics*, pages 9–18, 2014. 13
- [162] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. C. Berendsen. Gromacs: Fast, flexible, and free. *Journal of Computational Chemistry*, 26(16):1701–1718, 2005. 45
- [163] VideoLan. x264. Online, last accessed 22/04/2022. <https://www.videolan.org/developers/x264.html>. 70
- [164] VideoLan. x265. Online, last accessed 22/04/2022. <https://www.videolan.org/developers/x265.html>. 70
- [165] C. Vogel and J. Pleiss. The modular structure of ThDP-dependent enzymes. *Proteins: Structure, Function, and Bioinformatics*, 82(10):2523–2537, 2014. 154
- [166] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 96

- [167] E. Webb, editor. *Enzyme Nomenclature 1992. Recommendations of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology and the Nomenclature Classification of Enzymes*. Academic Press, 1992. 53
- [168] D. Weiskopf. *GPU-based interactive visualization techniques*. Springer, 2007. 8
- [169] F. W. Weymouth. Visual sensory units and the minimum angle of resolution. *Optometry and Vision Science*, 40(9):550–568, 1963. 13
- [170] O. Wiedemann, V. Hosu, H. Lin, and D. Saupe. Foveated Video Coding for Real-Time Streaming Applications. *Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6, 2020. 126
- [171] L. Xiao, S. Nouri, M. Chapman, A. Fix, D. Lanman, and A. Kaplanyan. Neural supersampling for real-time rendering. *ACM Transactions on Graphics*, 39(4), 2020. 106
- [172] Y. Xiong, J. Esquivel-Rodriguez, L. Sael, and D. Kihara. 3D-SURFER 2.0: Web platform for real-time search and characterization of protein surfaces. *Protein Structure Prediction*, pages 105–117, 2014. 46
- [173] C. Xu and J. Prince. Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*, 7(3):359–369, 1998. 30
- [174] L. Yang, D. Nehab, P. V. Sander, P. Sitthi-amorn, J. Lawrence, and H. Hoppe. Amortized supersampling. *ACM Transactions on Graphics*, 28(5):1–12, 2009. 105
- [175] Z. Yu. A list-based method for fast generation of molecular surfaces. *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 5909–5912, 2009. 18
- [176] A. Zaks and A. M. Klibanov. The effect of water on enzyme action in organic media. *Biological Chemistry*, 263(17):8017–8021, 1988. 44
- [177] A. Zare, A. Aminlou, M. M. Hannuksela, and M. Gabbouj. Hvc-compliant tile-based streaming of panoramic video for virtual reality applications. *Proceedings of the 24th ACM International Conference on Multimedia*, page 601–605, 2016. 126
- [178] Y. Zhang and J. Skolnick. Scoring function for automated assessment of protein structure template quality. *Proteins: Structure, Function, and Bioinformatics*, 57(4):702–710, 2004. 56, 57
- [179] S. Zhukov, A. Iones, and G. Kronin. An Ambient Light Illumination Model. *Eurographics Workshop on Rendering*, pages 45–56, 1998. 37