Context-Aware Techniques for Visualization

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

Mike Eißele

aus Esslingen am Neckar

Hauptberichter:	Prof. Dr. rer. nat. T. Ertl
Mitberichter:	Prof. Dr.-Ing. D. Fritsch
Tag der mündlichen Prüfung:	29.07.2010

Institut für Visualisierung und Interaktive Systeme
der Universität Stuttgart

2010

Contents

Abbreviations and Symbols

Abbreviations

API	Application Programming Interface	AR	Augmented Reality
AWS	Augmented World Schema	AWM	Augmented World Model
CNC	computerized numerical control	CPU	Central Processing Unit
CRM	Customer Relationship Management	DTP	Desktop Publishing
DOF	degrees of freedom	DVR	direct volume rendering
e.g.	exempli gratia (for example)	ERP	Enterprise Resource Planning
etc.	et cetera (and so forth)	FLOPS	floating point operations per second
fps	frames per second	Gbit/s	gigabits per second
GB/s	gigabyte per second	GLSL	OpenGL Shading Language
GPRS	General Packet Radio Service	GPS	Global Positioning System
GPU	Graphics Processing Unit	GUI	Graphical User Interface
HMD	head-mounted display	HLSL	High Level Shading Language
i.e.	id est (that is)	IP	Internet Protocol
IVR	indirect volume rendering	kB	kilobyte (1024 byte)
kbit/s	kilobits per second	KML	Keyhole Markup Language
LAN	Local Area Network	LCD	liquid-crystal display
LIC	Line Integral Convolution	MB	megabyte (1024 kB)
Mbit/s	megabits per second	mm	millimeter
MP(ixel)	megapixel (sensor resolution)	n/a	not applicable
OCR	Optical Character Recognition	OLE	Object Linking and Embedding
PC	Personal Computer	PDA	Personal Digital Assistant
pixel	picture element	POI	Point of Interest
RFID	Radio Frequency Identifier	RGB(A)	red, green, blue, (alpha)
RTT	round trip time	SDK	Software Development Kit
SIFT	scale-invariant feature transform	SIMD	Single Instruction Multiple Data
SG	Scene Graph	SVG	Scalable Vector Graphics
USB	Universal Serial Bus	VR	Virtual Reality
vs.	versus	WLAN	Wireless LAN

Symbols

$\otimes$	convolution	$\pm$	plus or minus
Θ	step function (Heaviside)		

Abstract and Chapter Summaries

Abstract

Visualization of data becomes increasingly important as the amount of generated data, captured or calculated, already reached an enormous extent and often cannot be interpreted without adequate visualization techniques. The development and invention of novel methods to present data in order to gain an in-depth understanding of data is still, and will most likely long be, an active field of research. This results in a permanent growth in the variety of data types and visualization techniques, raising the problem of choosing an adequate technique that visualizes the considered data appropriately. Even more difficult is the configuration and fine-tuning of visualization parameters to match characteristics of the considered raw data.

Therefore, this thesis focuses on context-aware techniques to address these non-trivial challenges of implementing, finding, selecting, and configuring suitable visualizations for arbitrary, heterogeneous data. Suitability is thereby strongly influenced by the situation of real-world (location, device, etc.) and virtual (data format, user preference, etc.) attributes, in summary referred to as context. The presented research contributes to various aspects for the development of a context-aware visualization framework and its application to realize context-aware visualization techniques. The framework is built on top of the distributed Nexus system, which provides spatial world models for mobile context-aware applications, and is designed to efficiently support mobile client devices.

Nexus provides fundamental functionality to query context information that is extended by this thesis with novel techniques to acquire and process context information on mobile devices. In particular, optical image sensors are used to capture features of the physical environment which are further processed to derive simple context information. To enrich this knowledge about the actual context, a novel interactive approach is proposed where context information is used to guide users in capturing additional context features. The aforementioned framework analyzes all relevant context data—originating either from the Nexus system or from client-attached sensor technology—and performs a XML schema matching to determine appropriate visualization techniques. For interactive renderings of the mostly huge amounts of raw data, graphics-hardware friendly representations have to be generated, again dependent on the context.

Finally, research on core context-aware visualization techniques is presented. Many common interactive visualization methods—e.g. color coding, volume rendering, stream lines, etc.—are easily integrated into the framework. Special emphasis is put on the development of interactive context-aware adaptation leading to improved or even new visualization approaches that are evaluated based on various prototype applications.

Overall, this thesis presents novel solutions to implement context-aware visualization that optimally present the user's data of interest, according to various context aspects.

Chapter Summaries

Chapter 1: Introduction

A detailed introduction into the topic of this thesis is presented, including a motivation and the goal of this work. It is highlighted that visualization is getting increasingly important which raises one of the upcoming key challenges in data visualization, addressed in this thesis: The variety of available techniques and their countless configuration options makes it a hard and time-consuming process to select and adjust the most appropriate visualization approach, even for expert users.

It is argued that context awareness applied to visualization can eliminate or at least alleviate this problem. Special focus is placed on visualization applications for mobile scenarios as context-aware behavior is essential for future smart mobile devices. Before entering a discussion about context-aware systems, an overview of different definitions of the term *context* are reviewed and summarized to establish an interpretation valid for this thesis.

Context-aware applications are finally introduced as an extension to so-called location-aware services with the addition that context aspects—other than location—are evaluated to influence application behavior. The chapter closes with examples of existing context-aware applications in order to demonstrate their potential for future applications.

Chapter 2: Context Management

This chapter elaborates on concepts and techniques for context management including: acquisition, interpretation, storage, and provisioning of context data. Existing framework solutions for building context-aware applications that offer support for the aforementioned tasks are reviewed. Based on this, Nexus (Spatial World Models for Mobile Context-Aware Applications) is selected as an underlying framework to enable techniques for context-aware visualization, proposed in this thesis.

A detailed discussion on Nexus highlights its fundamental concepts and gives an overview of the Nexus layers and services. Nexus' central augmented world model (AWM), a federated data source that is accessed by data providers and consumers, is introduced. Furthermore, the chapter details how visualization data, namely 3D geometry information, can be integrated into the existing data model of Nexus.

In general, information regarding the context is often not easily available; thus, sophisticated hard-/software is required in order to acquire sensor data. Although this non-trivial task is mainly handled by Nexus, it is still important to present the utilized underlying technology. Countless sensor technologies exist, but not all are adequate for practical context-aware applications. External, i.e. infrastructure-based technology, and local, i.e. client-device embedded sensors, are described with regard to context information acquisition.

An even more important component of context management is the reasonable interpretation of sensor data in order to generate context information, which is, in a sense, higher level information. Examples of different sensor data are shown together with concepts for meaningful

interpretations in order to generate context knowledge. Specifically, interpretation of orientation information and evaluation of captured images using optical character recognition are discussed.

Mobile devices are the primary use case for context-aware applications but suffer from many limitations, especially the lack of embedded sensors to acquire potential context data. On the basis of object recognition, this chapter proposes a novel approach to overcome some of these limitations by an interactive context refinement, which is demonstrated via a prototypical implementation.

Chapter 3: Computer Graphics and Visualization

Before addressing the demands of context-aware visualization, basic technology and concepts of computer graphics and visualization are introduced in this chapter. Special emphasis is placed on graphics hardware related technology as most proposed techniques in this thesis are targeted for efficient graphics hardware utilization. A basic description of both, the graphics pipeline and the visualization pipeline are given.

First, the main stages of a rasterization-based rendering pipeline—as realized within modern graphics processing units (GPUs)—are illustrated and explained. Starting with geometry processing where the input geometry is transformed, the pipeline continues with the actual rasterization to convert geometry information to fragments, and ends with fragment operations that output the final result as pixels. The performance of GPUs, i.e. how fast they are able to execute the rendering pipeline, varies based on GPU type but also based on the utilized pipeline setup. Thus, measurement techniques and their results for performance characteristics of GPUs are presented that are essential when designing new visualization techniques. Also the basic low-level programming APIs for graphics hardware—namely OpenGL and DirectX—are presented. In the following, high-level interfaces are introduced that offer much more functionality and simplify GPU programming. Specifically, scalable vector graphics (SVG) and scene graphs (OSG) are mentioned as both are used in prototype applications, developed as part of this thesis.

Second, the most important concept for visualization is presented: the visualization pipeline. Analog to the rendering pipeline, all stages are illustrated and described accordingly. After the introduction of these basic computer graphics concepts, some additional approaches—relevant to this thesis—are introduced: the technology of virtual and augmented reality. Both approaches are strongly related to computer graphics technology and—to some extent—make use of context information. The chapter provides a definition of the terms and also shows how virtual and augmented reality are connected, depicting the meaning of mixed reality.

Chapter 4: A Framework for Context-Aware Visualization

Research and development of context-aware visualization requires a non-trivial framework to integrate various visualization techniques and cooperate with the Nexus platform. Based on existing technology for context-aware application development a framework is proposed to support context-aware interactive visualizations, especially for mobile devices. This chapter addresses all aspects of the developed generic approach, visualization techniques themselves are addressed in the following chapter.

Initially, related developments are discussed although sophisticated context-aware visualization approaches are not yet intensively considered in current research. Experiences from this related work, based on collaborations with many researchers in the field of context-aware applications and based on numerous prototype implementations, requirements that have to be fulfilled by a context-aware visualization framework are derived. As a result, a distributed framework design is proposed that supports arbitrary visualization approaches; thereby specific considerations are made to utilize computational power of graphics hardware and provide efficient support for mobile devices.

The chapter details framework support for context-aware selection and configuration of visualization techniques which is based on a hierarchical data schema definition, as available for Nexus' augmented world schema. Additionally, an approach to provide visualization even of unknown data formats utilizing the schema hierarchy is explained, which is an important feature when working with open data sources like Nexus.

Explicit support for mobile-device scenarios is highlighted with focus on two aspects: a device-dependent preprocessing and the optimization of network communication. It is shown how preprocessing depends on the client device by supporting three basic configurations: remote rendering, render local, and hybrid rendering. The proposed network communication optimizations are designed for mobile communication networks to overcome their intrinsic bandwidth limitation and latency compared to wired networks.

A discussion on approaches for context-aware user interaction closes the chapter. A generic system is presented that provides user interaction capabilities via orientation changes. The concept is shown and evaluated within different applications using a prototypical implementation.

Chapter 5: Context-Aware Visualization

In this chapter miscellaneous techniques integrating context-aware visualization approaches into the framework—introduced in Chapter 4—are shown using multiple examples of up to date visualization techniques. When depicting the relation of context-aware visualization to focus&context approaches, the proposed importance of context in visualization is further strengthened.

In contrast, also the advantages of artificial or even non-physical context are examined and evaluated. In particular, an application that allows to change the presented context information, i.e. physical vs. artificial, is presented. When visualizing data in combination with a presentation of its related context, the problem arises that visualization might hide important aspects of the context. The chapter addresses this conflict with two approaches: embedded visualization techniques that seamless integrate within the context presentation without occluding important features and zoomable visualization methods. Targeting mobile devices with their limited screen space, zoomable user interfaces are introduced as an instrument to freely adjust multiple detail levels of focus and context information.

For context-aware visualization a primary context aspect is the utilized display technology ranging from small-screen mobile devices to head-mounted displays. A detailed discussion on adaptation of images and videos presents multiple approaches to prepare information to match the properties of the output device, wherein display size and resolution are the most dominant aspects. However, also the available color depth of the output device may be limited which is

why halftoning techniques are demonstrated for low-color-depth displays.

The main part of the chapter presents a discussion on various alternative visualization techniques for specific data types and corresponding context aspects that are relevant to estimate the appropriateness of each visualization approach. In an overview several actual implementations of data visualization techniques are shown, focusing on scalar and vector data, and hints are given how to utilize context information to select and configure the most adequate approach in different situations. After selection, the final context-aware configuration of chosen visualization approaches is also described using different scenarios.

Chapter 6: Application and Results

Results of applications utilizing context-aware visualization approaches, as previously introduced are detailed and evaluated. In addition, aspects that are important when deploying context-aware technology to real-world scenarios are highlighted; thereby technical and ethical restrictions are considered.

The examined prototype applications are subdivided into multiple classes: mainstream applications for everyday tasks, systems to visualize simulated or measured scientific data, tools for manufacturing in the industrial sector, and an application in the increasingly popular field of geo-mashup. This diversity of presented example applications clearly depicts the potential of context awareness applied for visualization tasks. Within each class prototypical examples are presented and evaluated to show the benefit of a context-aware behavior.

Furthermore, the performance for selected applications is presented to show the overall potential of the proposed framework and context-aware visualization prototypes. Thereby, mobile device scenarios are explicitly considered and evaluated to motivate future research and development of advanced data visualization on smart mobile devices.

Chapter 7: Conclusion

The final chapter summarizes the developed concepts and the benefits of context-aware visualization. Topics for future work are given that will enable improved visualization approaches with better utilization of context information, so that users will benefit even more of the advantages of visualization.

In order to motivate future research on context-aware visualization techniques a sketch of a coarse workflow for the development of context-aware techniques is introduced. Although it reflects only basic steps, the workflow may serve as a good starting point for researchers and developers that are interested in working in the field of context-aware visualization.

Kurzfassung

Die Visualisierung von Daten wird immer wichtiger, da die Menge an Daten, ob gemessen oder berechnet, bereits enorme Ausmaße angenommen hat und oft nicht ohne geeignete Visualisierungsverfahren analysiert werden kann. Die Entwicklung neuartiger Methoden zur Darstellung von Daten, die es erlauben, ein tieferes Verständnis zu erlangen, ist noch immer ein aktiver Forschungsbereich und wird es mit hoher Wahrscheinlichkeit auch weiterhin sein. Somit entsteht eine permanent wachsende Vielfalt an Visualisierungsverfahren für unterschiedliche Datenformate, wodurch insbesondere das Problem verschärft wird, ein geeignetes Verfahren auszuwählen, welches die zu untersuchenden Daten entsprechend darstellt. Noch problematischer ist die Konfiguration und genaue Anpassung der Visualisierungsparameter auf die Charakteristiken der zu Grunde liegenden Daten.

Diese Dissertation befasst sich daher intensiv mit kontextabhängigen Techniken, um Herausforderungen wie die Implementierung, Auswahl und Konfiguration von geeigneten Visualisierungstechniken mit Hilfe von kontextabhängigen Verfahren zu ermöglichen. Welches Verfahren sich wann eignet, hängt dabei stark von der aktuell vorherrschenden Situation (Kontext) ab, die durch reale Eigenschaften (Ort, Gerät, usw.) und virtuelle Informationen (Datenformat, Benutzereinstellungen, usw.) definiert wird. Die dargestellten Forschungsergebnisse dokumentieren den Fortschritt in vielen Bereichen, die relevant bei der Entwicklung eines kontextabhängigen Rahmenwerks für entsprechende Visualisierungsverfahren sind. Das System basiert auf der verteilten Nexus-Plattform, die Umgebungsmodelle für mobile kontextbezogene Systeme bereitstellt und mobile Anwendungen effizient unterstützt.

Die Nexus Plattform stellt Basisfunktionalitäten für die Verarbeitung von Kontextdaten zur Verfügung, welche im Rahmen dieser Dissertation durch neuartige Techniken für die Erfassung und Verarbeitung von Kontextinformationen auf mobilen Geräten ergänzt werden. Dabei werden optische Bildsensoren zur Erkennung von Merkmalen der physikalischen Umgebung eingesetzt, deren Ausgangsdaten weiterverarbeitet werden, um einfache Kontextinformationen zu ermitteln. Eine Anreicherung dieser Daten wird durch einen neuartigen interaktiven Prozess erreicht, bei dem wiederum Kontextwissen eingesetzt wird, um weitere Kontextmerkmale durch gezielte Benutzerführung aufzunehmen. Das zuvor erwähnte Rahmenwerk für kontextabhängige Visualisierung analysiert relevante Kontextdaten, die entweder von Nexus geliefert werden oder durch lokale Sensoren, und bestimmt passende Visualisierungsverfahren durch einen XML-Schema Abgleich. Für eine interaktive Darstellung der meist sehr großen Datenmengen sind Verfahren einzusetzen, die Graphikhardware zur Erzeugung der Darstellung effizient nutzen.

Schließlich werden Ergebnisse der Untersuchung und Implementierung von kontextabhängigen Visualisierungsverfahren präsentiert. Einige bekannte interaktive Methoden, wie beispielsweise Farbkodierung, Volumendarstellung, Strömungslinien, usw., werden exemplarisch in das Rahmenwerk integriert. Dabei wird insbesondere Wert auf die Entwicklung echtzeitfähiger Kontextanpassungsverfahren gelegt. Somit werden neuartige Visualisierungsansätze möglich, die auf Basis von prototypischen Implementierungen evaluiert werden.

Im Rahmen dieser Dissertation werden daher Lösungen vorgestellt, die es erlauben effiziente kontextabhängige Visualisierungstechniken zu realisieren, die eine optimale Darstellung unter

Berücksichtigung verschiedener Kontextaspekte leisten.

1 Introduction

Visualization has the goal to display information in order to get insight and a deeper understanding of the considered data. Research has already provided various techniques and methods to handle and present data originating from measurement or simulation. But still, users are required to have a deep knowledge of the data and available visualization techniques to be able to choose adequate approaches and handle the complex task of data analysis by visualization. As the amount of data structures and formats continues to increase, including related visualization approaches, the process of selecting an adequate visualization technique will be further complicated.

Moreover, once a specific technique is chosen, the resulting illustration strongly depends on configured visualization parameters. Most visualization approaches offer a rich set of options that influences the final output, probably resulting in unusable or even misleading conclusions when misconfigured. Thus, users are required to have in-depth knowledge of the data characteristics (i.e. examined phenomena), data format/structure, available visualization techniques, and visualization parameter options to generate adequate visualization results. Although trained and experienced users are able to perform the aforementioned tasks, still relevant aspects might exist that are unknown to them which might lead to suboptimal results. Resolution of the display device, available processor/graphics performance, or network connectivity are only some aspects that additionally influence the usability of visualization techniques, e.g., on limited smart mobile devices. Furthermore, the frequency and accuracy of manual visualization adjustments is limited and is, therefore, not applicable for fast changing conditions like location and orientation changes of a user operating AR applications.

Not only have the demands on end users continuously increased, but also the requirements of visualization systems. Complex 3D data is required to be presented using interactive techniques—that is techniques which can be adjusted with a near real-time feedback of the result—in order to understand spatial structures. The considered data has to be embedded and shown within its relevant context using multiple different data sources from various providers that have to be federated. Thus, isolated visualization of single datasets will become less important versus integrated, combined visualization approaches.

1.1 Motivation

The upcoming interests in context-aware and ubiquitous computing address a similar goal: supporting users by *automatically* triggering specific actions. Therefore, context awareness plays a major role in current research, especially in combination with smart mobile devices, and opens the field for new innovative applications that support users in their everyday life.

In this thesis the concept of context awareness is applied to the field of visualization; thereby, context knowledge is utilized to adapt visualizations accordingly in order to address the increasing demands on users and applications. A context-aware visualization framework will enable an automatic selection and configuration of adequate visualization techniques without the need of user interaction. New visualization approaches can easily be integrated and are utilized by the framework, if applicable. Therefore, such a framework helps users to make use of most recent advances in visualization without the need of expert knowledge. Furthermore, even fast changing context aspects like location or orientation can automatically be considered to adapt interactive visualization techniques.

When considering context information to control visualization, a multitude of context aspects is relevant, but still one of the most dominant context data is location, leading to location-aware applications. Especially users operating a mobile device benefit from location-aware behavior; as can be seen on mobile navigation systems where GPS receivers are used to automatically update the user's current location. Consequently, mobile devices are a primary platform to deploy context-aware applications including visualization which have to be chosen and adapted according to device specific capabilities, probably unknown by users. The context-aware visualization framework developed within this thesis therefore integrates specific support for mobile devices allowing adequate data visualization even on small screen mobile devices.

1.2 Definition of Context

The word *context* is a loaded term and used in many different occasions. Most encyclopedias define context similar to Wiktionary's[1] definition:

> *The surroundings, circumstances, environment, background, or settings which determine, specify, or clarify the meaning of an event.*

One way to interpret this generic definition specific to the field of computer science is to replace the term *event* with *data value*. Thus, in computer science context describes additional information about data entities, which might be position, time, or any other related attribute. In this thesis various parts of the entire context information—referred to as *context aspects*—are considered, although often only a subset is interpreted simultaneously.

Further definitions of context can be found in research, some are different but most are in coincidence. B. Schilit et al. name three important context aspects: where you are, who you are with, and what resources are nearby [179]. A.K. Dey defines context as any information that

[1] Wiktionary, the free dictionary: http://en.wiktionary.org

describes the situation of an object [46] which is similar to the definition given in the previous paragraph. G. Chen and D. Kotz give a similar definition of context in their survey report [35], but add the detail that context aspects which are interesting or relevant to users are also included.

Some research further classifies parts or aspects of context in terms of importance, e.g. A.K. Dey and G.D. Abowd in [48] and later C. Becker and D. Nicklas in [15]. The latter used the classification for data organization and management which is realized as part of Nexus, a framework for context-aware applications [38]. They considered identity, location, and time of an entity as primary attributes for indexing context information. Note that A.K. Dey and G.D. Abowd also include *activity* (to describe what is occurring in a situation) in the primary context-attribute class. Remaining context aspects are summarized as secondary context.

High-Level Context

A further way to organize context information is the introduction of a hierarchy. Context aspects that can directly be captured via physical sensors are grouped as *low-level context* or *observable context*; whereas information or knowledge that is derived from such sensor-based values is termed *high-level context* or simply *situation*. Thereby, *situation* might in fact describe events like: "A meeting takes place in room 2.3, participants are Mr. A, Mr. B, and Ms. C." which is not observable by a single physical sensor. This definition is explicitly used in Nexus, but also other researchers share this model [123].

1.3 Context Awareness

Systems and applications that *act on* or *change* their behavior based on perceived context aspects are *context-aware*. Thus, these systems are aware of their environment and can automatically react to changes. An equivalent definition is given by B. Schilit et al. in [179] and A.K. Dey in [46]. G. Chen and D. Kotz provide an extended definition of *active context awareness* and *passive context awareness* wherein the first category correspond to the previously mentioned definition. In contrast, *passive context awareness* describes a behavior where applications use context information to simply *show* updated values and do not trigger actions or change behavior, e.g. printing current GPS coordinates.

However, in practice this differentiation cannot easily be made—imagine a GPS device is capable to signal the arrival at specific coordinates and, thus, is acting *active context-aware*—so previous definitions of context awareness included both *active* and *passive* context awareness. Likewise, in this thesis no separation is made between active or passive; *context awareness* includes both categories. In fact, context-aware visualization techniques are researched in this thesis that adapt or even change dependent on context information, e.g. a highly inaccurate GPS position might be displayed differently compared to exact location information.

Context awareness, as used in this thesis, further includes *content awareness*. Processing data is said to be *content-aware* when the processing varies depending on the content of the data itself. In contrast, generic context awareness would further include other available information, not just the data itself, e.g. time, date, user, device, etc. Therefore, the definition also includes

location awareness, which can be interpreted as context awareness with the restriction to location information only.

1.4 Context-Aware Systems and Applications

Although context often does not explicitly provide additional information, it might be important to understand and interpret the considered data or information. E.g., the importance of context information to support the human visual system is proposed by A. Oliva and A. Torralba in [151]. They show that in some configurations humans can only interpret visual information if the related context is also presented. Applications that respect this fact are often termed as focus&context approaches, further detailed in Section 5.1. In addition to a visual representation of the context an automatic interpretation leads to context-aware approaches.

Despite a small number of commercial solutions, existing examples of context-aware systems and applications are mainly found in research as prototypical implementations. An overview is given in survey articles [35] and [10] including references therein. Basic examples are location-aware applications where, e.g., mobile users move in a defined environment and—dependent on the location—different information is revealed.

The GUIDE project [41] at the University of Lancaster primarily developed by N. Davies, K. Cheverst, and K. Mitchell utilize this functionality to realize and deploy a tourist information system for Lancaster. One of the earliest context-aware research applications is presented by R. Want et al. [201]. Their application tracks users and routes incoming phone calls to the nearest telephone, dependent on the user's location. More complex systems are, e.g., the AWARE HOME project at Georgia Institute of Technology [1] where various research groups work together on context-aware computing. The context-aware cell phone project, developed by R.W. DeVaul and S. Dunn, is part of the MIThril project [45] with the goal of a context-aware mobile phone capable of sensing the current situation to, e.g., select an appropriate setup profile when users are within restaurants, cinemas, or participate in a meeting. The focus of Aura [77] is to utilize context-aware computing for minimizing user interaction and with it to minimize user distraction. This project envisions a ubiquitous computing scenario where users stay in contact via their personal *aura*, a context-aware information and communication interface.

Specific context-aware applications are also researched in the Nexus project [38]. As Nexus targets on the development of a framework, several context-aware applications are considered in order to apply the framework and evaluate the envisioned concepts. The smart factory [206, 207] is one of these application scenarios in which prototypes developed in this thesis were integrated. The goal of a smart factory is to use context information to maintain a model of up-to-date status information of resources and entities within a real-world factory. This enables improved process optimization to increase productivity and lower costs based on actual data.

Most well known commercially available context-aware applications are GPS-based navigation systems. Numerous (mobile) devices are available that sense current location and update a map visualization accordingly, thus acting location-aware. As the current location is automatically known to the system, integrating routing support to find a route from current location to specific destinations is simplified. Even though mobile navigation software is already well

known, current research continuously provides new innovative enhancements: S. Möser et al., e.g., propose a context-aware presentation for navigation routes to emphasize the map region along the route via non-linear perspective projections [142].

2
Context Management

Recent advances in research exhibit an increasing effort to utilize context information for processing and presenting data. In the field of information visualization the term *focus&context* is well known and characterizes illustration techniques that present data in combination with its related context data. Focus&context therefore often combines different data types that show distinct aspects so that users are able to see, e.g., the data's spatial relation. Maps are a good example where layouts of houses, spatial regions, and streets are combined into a single visualization to see their spatial relation, making it easier to be interpreted by users.

Providing an open framework for storing, processing, and managing such arbitrary context data is the primary goal of the Nexus project. The general approach is to use context information for enabling novel applications. Thereby the notion of context data is widened compared to focus&context techniques and, in general, unrestricted. Enhancements of this framework with further concepts to realize novel context-adaptive visualization techniques are addressed in this thesis.

2.1 Nexus: A Framework for Context-Aware Applications

Acquisition, (pre-)processing, and storage of arbitrary context data can easily be provided by Nexus, a framework for mobile context-aware applications [38, 39]. The open system offers the possibility to query basic context information or even an estimation of a high-level situation description. The concept allows the integration of any data provider and any data consumer. The support of quality metrics further helps to weight the received context data. Using Nexus as an underlying service enables access to a huge variety of data—e.g. a user position, lighting conditions, available hardware, etc.

Nexus stands for the Collaborative Research Center 627 with the title: "Nexus: Spatial World Models for Mobile Context-Aware Applications", funded by the German Research Foundation. It represents projects that address various aspects of context awareness and context-aware applications, ranging from philosophical acceptability evaluations over semantic context models to security/privacy aspects in context-aware systems. One of the most important features of Nexus is the federation component which provides a common world model composed of arbitrary data providers possibly providing same data types for different or overlapping regions.

However, numerous similar projects exist that develop a framework for context-aware sys-

tems, all of them having advantages and disadvantages. An extensive survey on these context-aware frameworks is presented by M. Baldauf et al. in [10]. One of the earliest frameworks—stick-e documents—is proposed by P.J. Brown [27] and J. Pascoe [157]. So-called stick-e documents are attached to particular context constellations C, e.g. a specific GPS-based position, and displayed when the current context matches C. In the context toolkit proposed by A.K. Dey et al. [47, 175] a decentralized system acquires and processes data from remote sensors (widgets), thereby the framework is designed to isolate applications from the task of context management, similar to GUIs that abstract from user interface presentation. JCAF (Java Context-Awareness Framework) is a Java-based service-oriented API to develop context-aware applications, proposed by J.E. Bardram [11]. The event-based middleware supports distributed cooperating context services and, in contrast to most aforementioned frameworks, specifically addresses security and privacy issues. SOCAM a service-oriented context-aware middleware is primarily developed by T. Gu and H.K. Pung [84, 163]. It integrates an interpreter server that is capable of inferring deduced contexts, based on a context knowledge database. Thereby, the knowledge database is represented as multiple ontologies for different sub-domains (home, vehicle, person, etc.).

Although many different frameworks for context-aware application development exist, in this thesis we focus on Nexus as a basic underlying context-aware framework. Therefore, in the following a detailed introduction of Nexus and its concepts is given.

2.1.1 Conceptual Overview

The basic approach used in Nexus to support context awareness is to link context information with data entities to build an Augmented World Model (AWM). Thereby, data entities of the AWM might be captured real-world information, like sensor values, or virtual (non-real) information, like web-based information stores. Type, structure, and object definition of the data contained in the AWM is given by the Augmented World Schema (AWS), which is a hierarchical XML schema. As Nexus is designed to be an open, distributed system—in terms that everybody can contribute to the system, just like in the World Wide Web—a component to provide a unified data access for applications is required. In Nexus the so-called Federation Layer is responsible to control and simplify access to distributed data sources.

Figure 2.1 shows the structure of Nexus with its three-layer architecture; note that communication between all components is based on a network protocol to enable highly distributed setups. Applications that make use of Nexus are included in the topmost layer, the middle layer is defined by the AWM, and the bottom layer reflects the raw information or data. Thereby applications are end-user applications, or client programs, but also services that are based on the Nexus technology and provide further support for context-aware applications.

2.1.2 Augmented World Model

Sensors are used to capture real-world quantities. The wide range of sensor technology reaches from temperature sensors over pulse-frequency detectors to face recognition and many more. All this data provided as raw sensor information can be seen as a digitization of the physical world and is therefore represented as a part of the AWM. In addition, virtual data—like digital

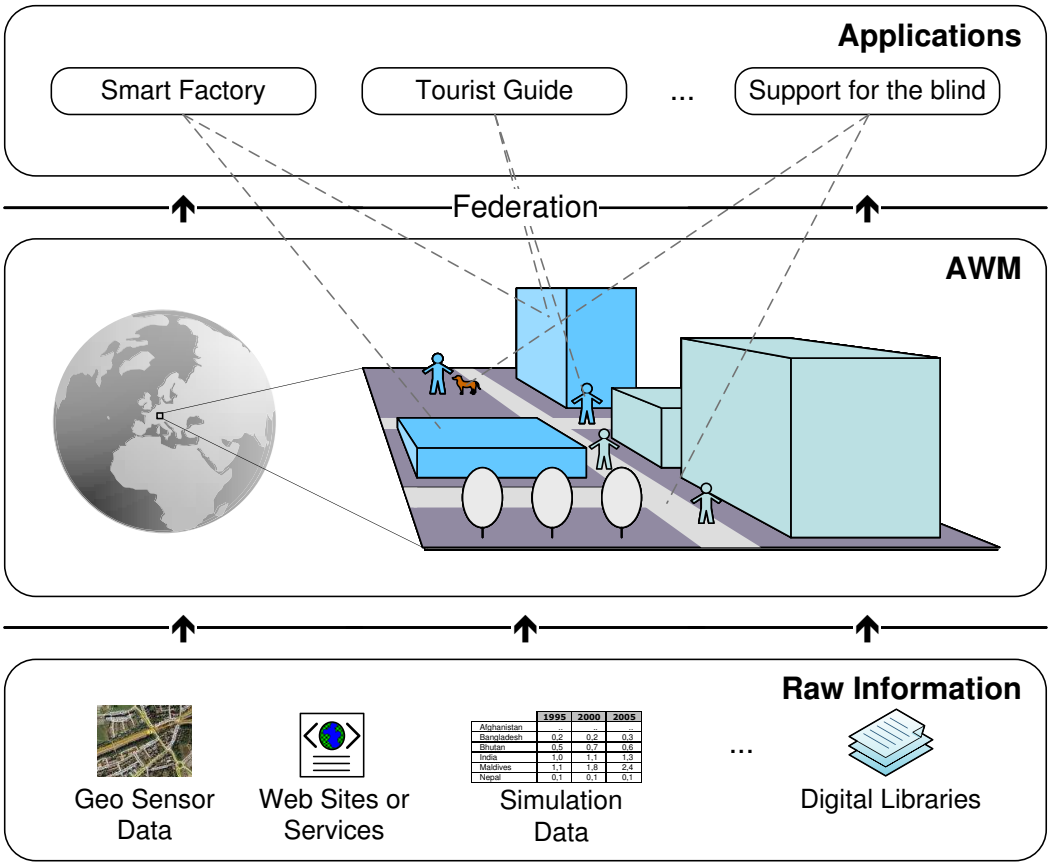


Figure 2.1: Conceptual overview of the Nexus layers.

information stores, internet pages, or digital libraries—is also embedded into the AWM and enriched with context information.

Techniques for storage and management of huge amounts of data are already researched in the field of databases. In contrast, the AWM supports primarily the management of context information and is built on top of an underlying database. In fact, the AWM consists of an open, distributed network of context servers that are responsible to store context data, possibly specialized for specific context aspects (e.g. highly dynamic location information). Access to data entities based on their contextual information, e.g. position, status, situation, etc., is therefore efficiently supported. Furthermore, the supported data structures and data types are not limited to fixed table structures of type definitions, which is a strong requirement for an open system. Extensions for arbitrary data and object types are supported to allow an integration of all possible information or data elements available, which is further detailed in Section 2.1.4.

2.1.3 Nexus Services for Context-Aware Applications

An important aspect of Nexus, to support the development of context-aware applications, is the integration of context-aware services that are based on the availability of contextual information

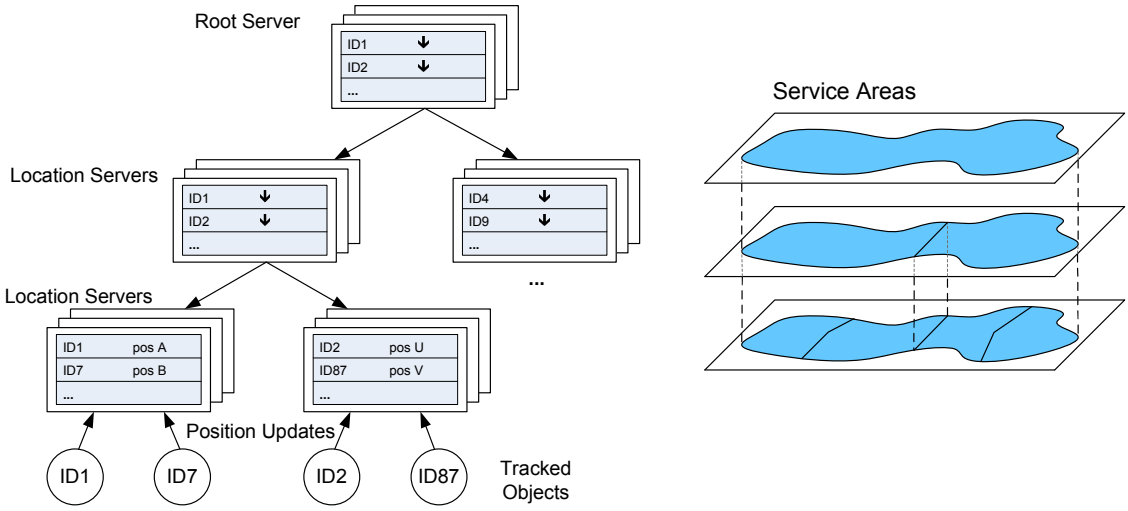


Figure 2.2: Architecture of the Nexus LocationService.

related to data entities. These services make use of context information to provide novel functionality, often not available in non-context-aware scenarios. In the following, a short description of the most important services is given.

LocationService For efficiency, the location of mobile, possibly fast moving objects are tracked via LocationServers. A hierarchical organization of multiple LocationServers is used to implement the LocationService. Context-aware Nexus applications can use this service to, e.g., query positions of objects within an area or retrieve n-nearest neighbors. The hierarchical architecture of the LocationService is illustrated in Figure 2.2, more details can be found in [125].

EventService Similar to database triggers, events are activated if predefined constellations within the AWM occur. This service is subdivided into an ObservationService to observe context information and its changes in the AWM and the NotificationService which is responsible for distributing the event messages. Applications can register events by providing a predicate to the EventService and receive event messages whenever the predicate resolves to true. This way, applications are able to react on events like: "a friend is within the surrounding."

Geocast A well-known example for context-aware application scenarios is the possibility to communicate with spatially addressed receivers. This so-called geographic communication is realized utilizing the available context information of the AWM. Users and applications can define a georeferenced area and send messages to it. Thereby, the message will only be delivered to clients within the defined area. Geonodes are responsible for the transmission of geocast messages by forming an overlay network to optimize communication costs.

NavigationService Advanced support for location-aware applications—where context-data is primarily location—is available through this service. Based on it, tasks like navigation, wayfind-

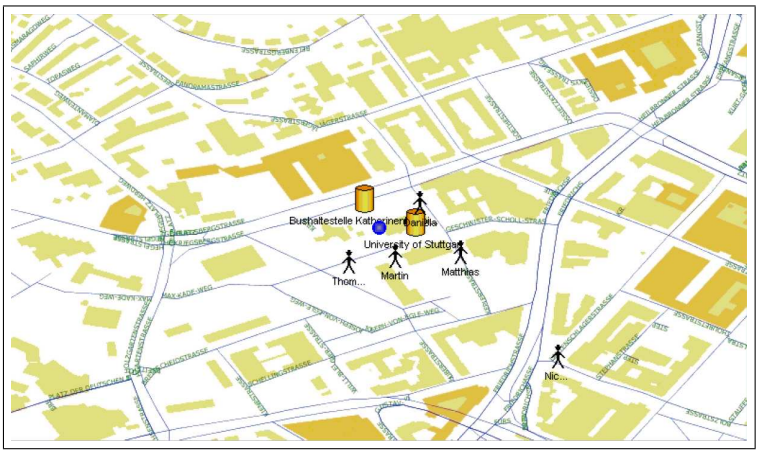


Figure 2.3: Region of the AWM, rendered using the Nexus MapService.

ing, route planning, etc. can easily be implemented. The service supports, e.g., queries for shortest-path routes and provides a sorted list of geographic way points that describe the route.

MapService A simple 2D mapping service that is able to generate static SVG- or bitmap-based maps of a requested region. A setup given as part of the query allows configuring the graphical representation of the individual data types. The service offers a simple mechanism to render regions of the AWM for non-interactive applications. An example illustration of a rendered AWM region is depicted in Figure 2.3.

Select3DService Previous implementation of Nexus components—applications and services—were targeted for the 2D domain. In order to support applications that are capable to handle and probably require 3D coordinates, without the introduction of incompatibilities, a translation service was developed to support 3D spatial queries. The service maps 3D queries to 2D queries and forwards these to the Nexus federation. The returned result is then cropped according to the application's 3D query.

2.1.4 Data Formats

A major challenge in an open system is the definition of data structures and data types. In Nexus, data types are hierarchically defined using a XML schema which is based on simple types like string, integer, or float. Therefore, data providers and applications can easily extend the available data definitions to introduce new, possibly complex, data types.

The AWM of Nexus unifies real-world and virtual data; therefore, definitions for both are required. In general, arbitrary data can be integrated in the AWM thus a definition for every possible data type is required. This requirement is impractical and cannot be achieved. To overcome this limitation, Nexus strongly makes use of already available data definitions. This way image formats, address data, or road networks are supported using existing standards. Furthermore, application specific data that can only be interpreted by very few applications and does not

need to be processed within the framework is not represented in Nexus. For such data objects, Nexus stores a light-weight object with related context information—e.g. status information, data format, or position—and an external reference to the data values in form of a URL. This way, compact, efficient binary data formats are also supported to allow inclusion of huge data elements that cannot efficiently be represented using XML. Results of simulations or measurements, like 3D vector fields, easily reach a data size of hundreds of megabytes and are therefore stored externally using a well-known binary format. Data format, field dimensions, position, and other context information are stored directly in context servers of Nexus.

Storing data externally in its native format has the advantage that a XML format description for representation in Nexus is not required and the data can be stored in binary format often consuming much less space compared to a XML description. In contrast, processing, filtering, or evaluation of such data within the Nexus framework is not possible anymore since therefore Nexus would need to understand its structure. An integration of data formats into Nexus for important data types that have to be processed within the framework is thus necessary. One such important data format, concerning visualization and rendering of data, that has to be integrated is the description of 3D geometric models. As various techniques developed for this thesis work with 3D geometric data, a definition to store 3D data within the Nexus framework is developed in cooperation with N. Hönle, T. Schwarz, S. Volz, M. Kada, L. Jendoubi, and S. Bürklen [56].

XML-based 3D Geometry Data

XML-based standards for 3D geometry models are already available, but most of them are designed for single, small-scale models like the X3D format [199] or Collada [111]. The usage of 3D geometry models in Nexus is many-sided and may be used for: Consistency/accuracy evaluation, simulation input, visualization, model matching, etc. Therefore, requirements for all possible use cases have to be considered. A major drawback of the aforementioned formats is the lack of a georeferenced positioning. The Keyhole Markup Language (KML), used in Google Earth [80] to embed user-defined content, can be used to add this missing functionality to Collada via an additional XML-based file to define the geographic reference. A zip-compressed archive of a *.kml* file and Collada models is often stored as *.kmz*. Still, for Nexus applications the description of the geometric models using Collada is too limited for arbitrary applications as it is primarily targeted for rendering.

Early implementations of Nexus used Geography Markup Language (GML) [153] to define georeferenced 2D regions. The recent version 3.0 of GML now also supports a description of 3D geometry, is not limited to visualization applications, and is, thus, a natural technology to support 3D geometry in Nexus. However, GML is suitable to define geometry but lacks any possibility to specify material properties that are required for visualization. Because of this, an extension of GML 3.0 is developed in [56] and similarly solved in the recent development of CityGML [152]. As CityGML is still missing some features like advanced material properties or local coordinates, relevant in Nexus and for this thesis, a Nexus specific extension is required. However, by contributing to the CityGML design process concepts developed for Nexus can be integrated into the CityGML development.

Representation of 3D Models in GML 3.0

The specification of GML 3.0 foresees the definition of 3D geometry via so-called solids, i.e. volumetric objects. We extended GML's definition in order to support surface materials and point-attributes (e.g. normals). The data is integrated using list structures allowing to reference, e.g., a single material definition multiple times within one geometry model.

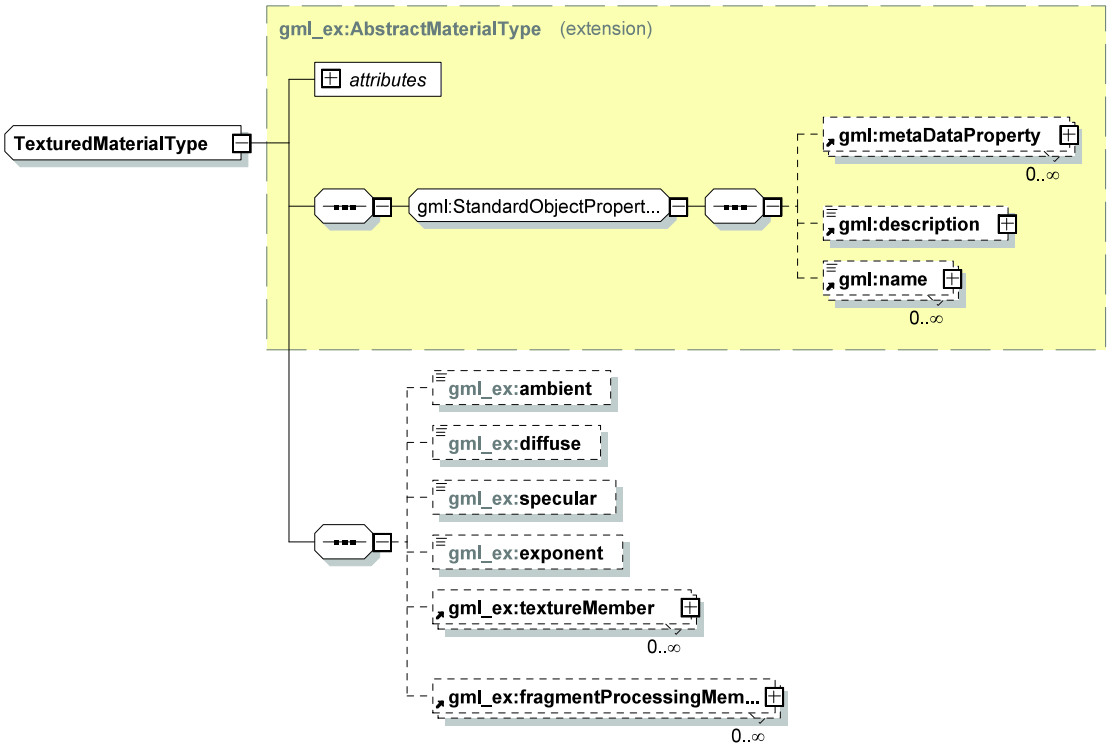


Figure 2.4: XML schema definition of the GML 3.0 extension for surface materials.

A single element of a material list is termed *TexturedMaterialType* and extents GML's standard object properties with attributes commonly used in computer graphics. A schematic view of the XML definition is given in Figure 2.4. The *ambient, diffuse, specular* attributes are color values (defined in red, green, blue portion); the *exponent* is a scalar value to adjust the specular reflection behavior of the material. Multiple *textureMember* elements are supported to define images that should be mapped onto geometries that reference the material definition. The function to combine all aforementioned material attributes to a final surface color is defined using the *fragmentProcessingMember*. Multiple alternative definitions of this element are possible to support different capabilities or qualities of applications that interpret this data. Thus, even algorithms—so-called fragment programs (Section 3.1)—can be defined that have to be executed for each displayed point of the surface in order to evaluate its color.

The schema definition for GML solids is extended with the type *TexturedSolidType* as shown in Figure 2.5. In contrast to standard GML solids, lists to store vector values—points, normals, or texture coordinates—are added. Within the elements *exterior/interior* these values are referenced

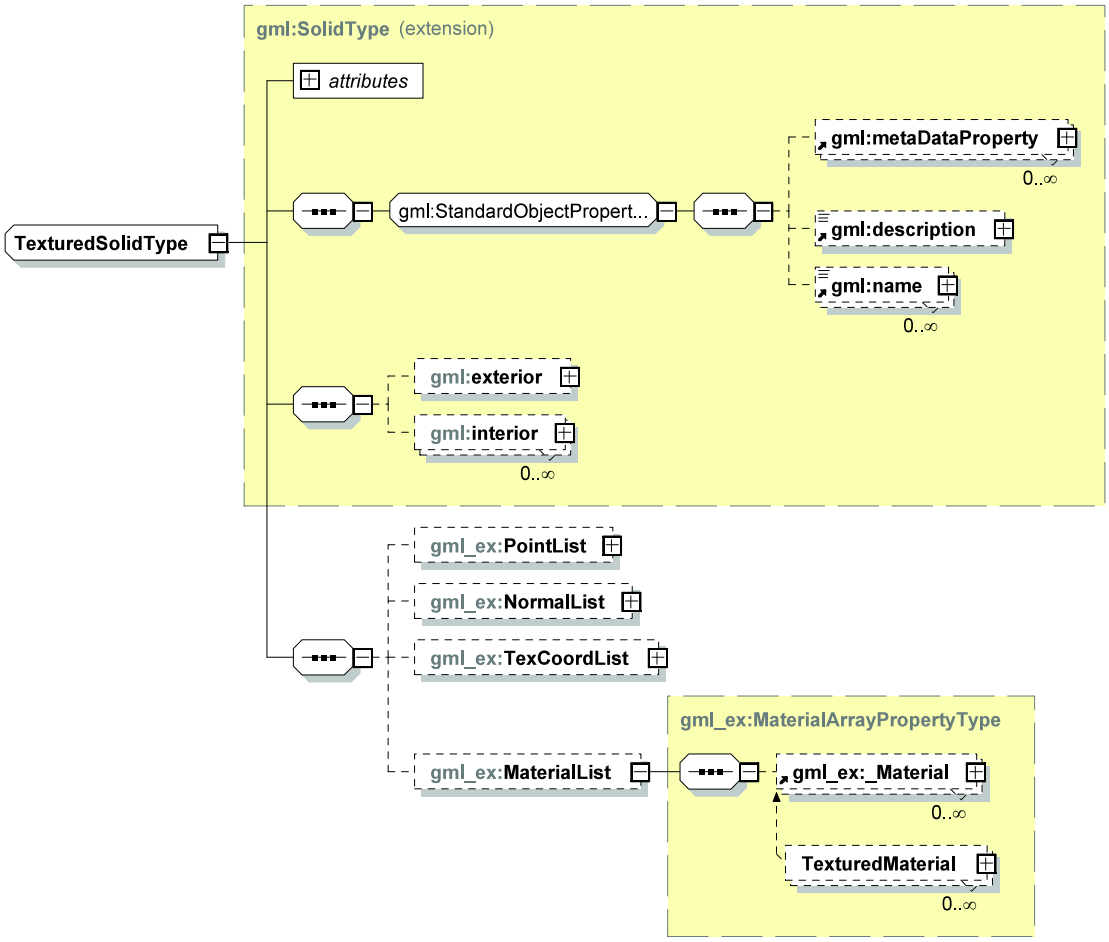


Figure 2.5: GML 3.0 extension for surfaces with referenced attributes and materials.

to define the geometry. Furthermore, a list of materials—as described above—can be added to specify the materials of the 3D geometry.

With these extensions a modeling of simple 3D objects is possible, but complex 3D geometry models are often defined hierarchically. Using multiple coordinate frames, embedded within each other, each subpart of the entire model is modeled using its local coordinate frame. The same concept of hierarchical coordinate frames is used in the KML format to define the local space of the referenced model, as mentioned before. In this case, an additional conversion from a geocentric to a Cartesian system is performed. GML has already integrated support for various coordinate-system types and different options to define transformations between these systems, but they are intended to be used for static definitions, like a new geospatial reference system. Because of this, coordinate-system definitions are stored in a separate XML file in GML. In contrast, geometry models in computer graphics make use of many local coordinate systems that are essentially only valid for subparts and are therefore naturally defined together with the model.

In [56] this problem is addressed by embedding the coordinate definitions in GML solids

definitions using the GML provided *metaDataProperty* tag. Listing 2.1 shows an excerpt of the XML definition of a solid including a local coordinate system.

```
<gml:Solid gml:id="Solid1"
   srsName="urn:x-nexus:1.0:coordinateReferenceSystem:817">
   <gml:metaDataProperty>
      <gml:GenericMetaData>
         <gml:DerivedCRS gml:id="NexusCRS817">
            ...
         </gml:DerivedCRS>
      </gml:GenericMetaData>
   </gml:metaDataProperty>
   <gml:exterior>
      ...
   </gml:exterior>
</gml:Solid>
```

Listing 2.1: Definition of a local coordinate system within a solid definition.

In computer graphics, coordinate transformations are mostly defined via 4x4 matrices using the concept of homogeneous coordinates (see Section 3.1). The GML specification includes only a basic structure to encode transformation methods and parameters where the applications themselves are responsible for correct interpretation. Therefore, a 4x4 matrix can be stored using a GML *valuelist* tag with 16 scalar values inside a standard GML *conversion* tag.

2.2 Context Acquisition

A fundamental part of context-aware systems and frameworks is the acquisition and interpretation of context information. Thus, some research of this thesis also contributes to this area, although the main focus is on usage and visualization of data with attached context information.

Sources for context data can be manifold: locally attached sensors, pre-acquired or simulated data stored in information databases, or knowledge-based systems. The Nexus framework unifies access to all these sources and is therefore the preferred context-data provider for all techniques developed in this thesis. The drawback is that Nexus introduces a noticeable latency which may be inadequate for real-time context-information processing, e.g. a fast changing position. In contrast, for *external* sensors that have to be accessed via a network interface, e.g. infrastructure-based location systems, the advantage of using their proprietary communication protocol is often marginal whereas the unified Nexus interface simplifies their usage.

However, dependent on the scenario, the latency when accessing context information via Nexus may be too high. Especially for applications, that are required to response at interactive rates, already a delay of $\sim$200 ms is strongly noticeable. Also, some sensors have to be locally attached, e.g. GPS receivers have to be with the mobile user, or are intrinsically local, e.g. application status or user adjustments. Using locally attached sensors naturally helps to reduce the latency but requires accessing and interpreting the raw sensor data without the support of Nexus. Therefore, different setups to acquire context data—local and external—are examined

and developed to support the research on context-aware visualization techniques.

In general, numerous sensors exist which may be used to acquire context data. In the following some sensors are discussed which are used in projects developed as part of this thesis. Also the often required interpretation of captured sensor values in order to make the raw data useable within applications is discussed.

2.2.1 External Sources

External sources of context-related data are providers that are not directly accessible, i.e. are not locally attached. A data communication protocol is used to query and retrieve data from those sources, for mobile scenarios often WIFI communication is used to access an external data service.

Infrastructure-based positioning systems are important external context providers. These systems make use of preinstalled infrastructure—like electromagnetic beacons, multiple cameras, or just GPS repeaters—to calculate and provide an absolute position of previously marked or trained objects. Such systems are mainly targeted for high-precision indoor tracking. In particular an infrared-based system from Advanced Realtime Tracking [4], the Intersense [101] making use of ultrasonic sound, and the ultra-wideband-based Ubisense [196] were used in projects of this thesis; thereby, all of the systems have advantages and disadvantages.

The A.R.T. system [4] offers high accuracy and real-time response but requires multiple specific cameras and during operation a line-of-sight to relatively large tree markers—infrared-reflective markers with a specific geometry to allow 6 DOF tracking. The operation volume is extensible via additional cameras, but areas larger than >30 m^2 are unusual; the minimal configuration supports an area of <2 m^2. Intersense IS900 requires ceiling-mounted SoniStrips that emit ultrasonic waves that are received by tracking targets; this implies that the target is also an active component. Accuracy is also high although slightly lower compared to the A.R.T. system, especially the update rate is lower and an initial delay of movements is noticeable. Common volume of operation is similar to the A.R.T. system; larger volumes of up to 140 m^2 are supported.

For large-scale tracking of objects Ubisense offers a system that divides the tracking volume into cells and therefore supports theoretically arbitrary large environments. Within each cell multiple ultra-wideband receivers have to be installed and connected to a server that computes locations of wireless tags. Ubisense claim a position accuracy of 15 cm; however, orientation information is calculated by differentiating and is therefore inaccurate.

Different classes of infrastructure-based systems allow a local evaluation of location without direct communication to infrastructure components; location of other objects cannot be determined and have to be queried from an external database. The global positioning system (GPS) is the most prominent representative for outdoor scenarios. In contrast, WLAN transponders are popular to realize indoor localization. Research prototypes [212] and commercial solutions [64] are proposed. A popular example is the Ekahau system that requires communication to an infrastructure-based server for location queries, even for self-localization. However, an implementation that does not require access to external data sources to evaluate a location seems possible.

Apart from object identification and tracking further context-information is provided by external services whereas most have their proprietary communication protocols, including the aforementioned tracking systems. Work in this thesis is therefore based on a framework for context-aware applications that abstracts system-specific formats and protocols to retrieve context information. Still, (wireless) network communication is used to access the data of the framework, thus it is classified as an external data source for context data.

2.2.2 Local Context Providers

Sensors that are directly attached, e.g. via serial or USB interface, to the client device and therefore offer low-latency, high-bandwidth information are important local context providers. Even common user input devices like keyboard, buttons, mouse, stylus, or joystick are local sensors that capture user intentions. Static context information, e.g. device type, user preferences, etc., can be pre-acquired and stored in a database on the client device and are therefore a local context provider.

Inertial sensors are a well-known example for local sensors. Most commercially available off-the-shelf inertial orientation sensors are designed for virtual reality (VR) applications and differ greatly in precision and price. For a prototypical evaluation an InertiaCube^2 from InterSense [100] is used, which is a combined device of three gyroscope, three accelerometers, and three magnetic field sensors. The sensor is designed for high-precision head tracking in VR and has, therefore, a very small form factor—roughly 30 mm^3—and weights only 25 g. However, due to its target for professional market the price is about $1700. In contrast, new innovative mobile devices [6] or input devices—like the controller of Nintendo's Wii[1]—integrate inertial sensors. Often inertial sensors are combined with an electronic compass to provide direction information in cardinal points. Although a compass itself would provide absolute directional information in most applications it is only used to reduce drifting of a differential orientation sensor and to stabilize its output.

As mentioned, GPS offers location information without communication requirements and is therefore the preferred location system for mobile applications, e.g. in navigation software. However, the provided accuracy is rather low (>5 m) and the GPS signal can only be received outdoors. To overcome this limitation, GPS repeaters are available that can be installed inside buildings to propagate the GPS signal into buildings, but still the accuracy is in the order of [m] while previously mentioned infrastructure-based systems offer [mm] accuracy. WLAN-based location systems reuse the often existing wireless communication infrastructure for localization. However, large-scale experimental measurements showed that the accuracy is strongly influenced by changes in the environment, e.g. closed vs. opened window, and is in average in the order of [m].

For marker-based object recognition radio frequency identifiers (RFID) are common, even for mobile scenarios as demonstrated by research prototypes of mobile phones that already embed RFID readers [176]. In principal, RFID tags are similar to infrared beacons as reading devices just receive an identification code, without information of distance or direction the signal origi-

[1] Nintendo Wii Remote Controller: http://www.nintendo.com/wii

nate from. Depending on the type of the tag different operating ranges are possible: passive tags often operate in the range of [cm], active tags may range up to 10-20 m.

In general, local context providers offer a much lower latency at a much higher data bandwidth compared to external sources. This is especially important for sensors that continuously provide huge amounts of data that have to be processed. Actually, low resolution cameras (640x480 pixels) that are embedded in, e.g., smartphones generate about 20 MB of data per second. Even if external sources exist that provide such information, then the available data rate of most wireless communication does not offer enough bandwidth to transfer the data in realtime. Consequently, interactive context-aware systems are required to support locally attached sensors to receive and interpret real-time context information. E.g., for some projects the computer-vision based AR-ToolKit [110] is utilized to calculate real-time marker location information based on a live video image, captured by a locally attached video camera.

For research on context acquisition this thesis concentrates on techniques based on: inertial sensing and optical sensors, i.e. cameras, discussed in the following sections. Further a method is shown how context knowledge is interactively enriched with additional aspects resulting in an object recognition/identification system.

2.2.3 Interpretation of Inertial Orientation Data

Second to location context, orientation information is often considered as a further aspect of context. Already J. Rekimoto [167] presents input interfaces on hand-held devices using sensor-provided orientation information of the mobile device to navigate through menus, although orientation is not interpreted as context. Many more similar prototypes are proposed: B.L. Harrison et al. [89] show the benefit of utilizing different sensors to capture gestures of the operator; a similar setup is described by K. Hinckley et al. [94]. J.F. Bartlett [13] shows an alternative input technique were the orientation of a hand-held device is used to scroll and navigate within an electronic photo album. Using device tilt gestures as a text input method is presented by K. Partridge et al. [156]. C. Verplaetse [197] presents a good survey on inertial proprioceptive devices. A. Schmidt et al. [180] show a simple hand-held prototype equipped with two mercury switches to recognize the orientation of the device.

Augmented and virtual reality systems often require orientation information for alignment of rendering or for interaction. However, only little research is done in adapting concepts of ubiquitous computing interface techniques to AR or VR systems, i.e. utilizing orientation information for interaction. D. Tan et al. [192] use some interaction techniques based on changing the orientation of augmented objects.

Using orientation information for context-aware control of applications and an evaluation of the concepts is proposed together with S. Stegmaier and D. Weiskopf in [60]. Orientation can be measured relative to a previous orientation or absolute to a given coordinate system. Relative orientation can be captured by inertial sensors that are based on, e.g., gyroscopes. For measuring the absolute orientation a coupling has to be established between the device and the reference coordinate system. Using a high-precision tracking system calibrated in reference coordinates can directly provide an absolute orientation including the absolute position but requires an external installation as discussed in Section 2.2.1. Compared to inertial-sensor-based systems this is a ma-

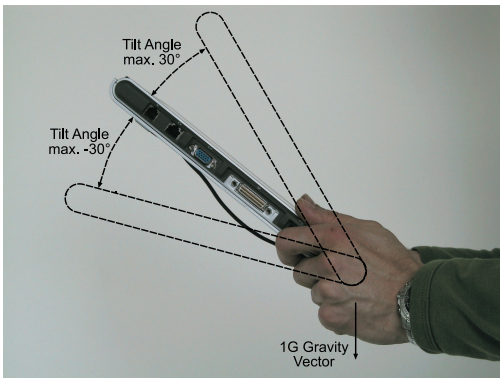


Figure 2.6: Maximum range of tilt operations.

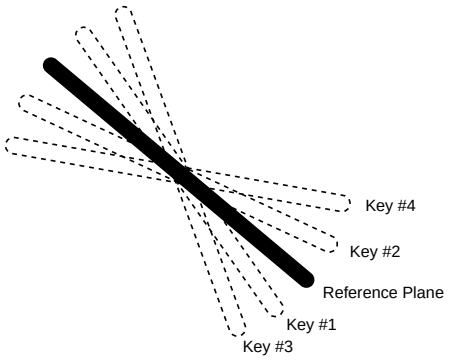


Figure 2.7: Defining different key events for different tilt angles.

jor deficit. The examples shown in [60] are therefore primarily based on orientation information only which allows a mobile operation without external installations.

Mapping Orientation to User Interaction

Changes in the device's orientation can directly be mapped to some common user interactions as proposed in [167]. The choice of this mapping greatly affects the user acceptance of the system because the operations performed when the orientation of the device is changed should always be obvious to the user and if possible dependent on the current situation, e.g. active application, user configuration, etc.

For mapping orientation, the angle of the device, also referred to as *tile angle*, must be known with respect to the user's reference orientation, as illustrated in Figure 2.6. The reference orientation, also referred to as *reference plane*, represents the default orientation of the device when the user does not apply any orientation changes for interaction. The setup of the reference plane is user specific and should be adjusted according to his working position and environmental lighting conditions. Obviously, a reconfiguration becomes necessary very frequently as users tend to change their working position from time to time. Therefore, an adequate solution is to setup a single button to capture the current orientation as new reference plane.

Another challenge for the system is to recognize when the user tilts the device but does not want to perform any user interaction. This happens in cases when the user stops working with the device and puts it down on a table or when the user changes his working position. The system might misinterpret this new orientation and will execute interaction events. To prevent this behavior the system somehow must recognize if the user tilts the device explicitly for user interaction. J. Rekimoto [167] suggests using a *clutch button* to activate the tilt functionality which allows utilizing tilting for user interaction only while the button is pressed (*clutch mode*). Secondly, a *lock mode* can be integrated where the mapping of orientation to interaction can be turned on or off. The clutch mode offers the additional benefit that each time the button is pressed the reference plane could be readjusted to the current device orientation. Therefore, the reference plane adjustment is embedded within the activation process of the user interaction via

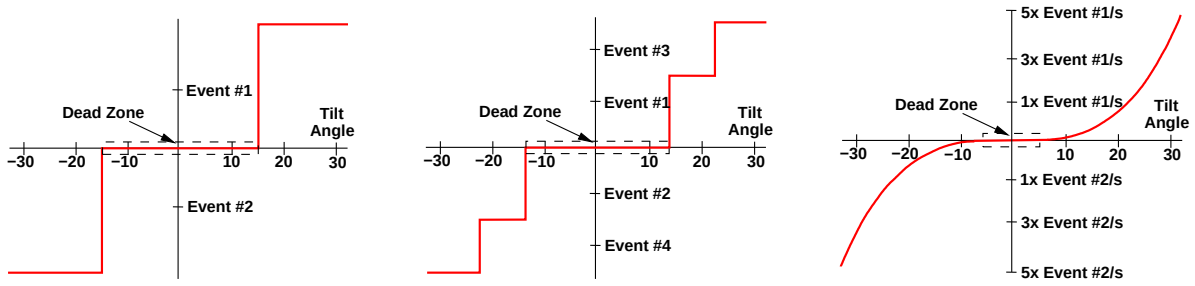


Figure 2.8: Different styles for mapping orientation to user interaction. A mapping for two events is shown in a), likewise b) shows a function to map multiple discrete events. A mapping for continuous events is depicted in c).

tilt gestures.

During prototype development it turned out that users did not tilt the device for more than about 30 degrees, thus interaction techniques that require more extreme changes in orientation were neglected. On the other hand as users cannot hold the device without slight changes in orientation a *dead zone* is used to prevent execution of events at orientations only slightly different than the reference plane as can be seen in the Figure 2.8. There are three possibilities to map orientation: a discrete, a partially constant, and a continuous mapping:

Discrete mapping uses a threshold on the tilt angle to recognize user interaction events. A typical example function to map the tilt angle to discrete user interaction events is shown in Figure 2.8a. This technique can, e.g., be used to simulate single key-press events or toggling of application states.

Partially constant mapping can assign different events based on different tilt angles. To decide when to measure the tilt angle, a timer can be used to measure the time when the orientation does not change significantly. If this timer exceeds a user defined threshold it is assumed that the user positioned the device to its desired orientation and the system measures the tilt angle. Another possibility is to store the angle where the user stopped the tilt gesture and starts to change the orientation back to the reference plane. This tilt angle is then used to decide which action has to be executed by using a mapping function as shown in Figure 2.8b. Figure 2.7 illustrates the device held at diverse tilt angles to trigger different events. This method can, e.g., generate a cursor down key-press event for slight tilt gestures and a page down key-press event for more serious changes in orientation.

Continuous mapping allows generating multiple events within a time period. The number of events generated depends on the tilt angle. The more the device is tilted the more events are generated. The typical mapping function used in this scenario is illustrated in Figure 2.8c. This mapping is useful to, e.g., choose elements within a list where the users can step through list elements or scroll window content, at varying speeds.

Using additional context information to select and adjust mappings of orientation to user interaction allows more sophisticated mapping techniques, especially if more information about the user's active applications and his preferences are known. In particular, AR applications can

benefit from more intuitive and supportive context-aware user interfaces as users have to interact with the real and virtual world concurrently. But also the entertainment sector may use this kind of user input to enable games like the dexterity puzzle or the well-known computer game Marble Madness by Atari[2].

Context Awareness based on Orientation

Compared to a direct mapping of orientation to user interaction, more elaborate interpretation of inertial sensor information allows deriving additional context information.

Most mobile devices are designed to be operated not just when holding still but also while in movement. Picking small graphical user interface (GUI) elements while standing still seems to be no problem for most users, but becomes a big challenge when walking. Sensing the orientation of the device with high precision allows the device's operational status to be recognized, i.e. if it is operated while moving or not. With this information the user interface can be modified to, e.g., include less functionality and larger control elements that are easier to tap. Furthermore, if the device experiences no movements at all—less than the natural shiver of a human hand—it must have been laid down. A device such as a mobile phone can then suspend most functions to save energy; since in general, it is operated in-hand. In contrast, if the sensor recognizes huge, rapid changes in orientation it can be assumed that the device is currently transported and that the heavy movements make it impossible to work with it. It can therefore just as well shut down and save power. Providing the possibility to sense an absolute orientation by adding a gravity sensor as described in Section 2.2.2 allows to add further functionality to the system. This simple enhancement allows to retrieve context information and to support the user by, e.g., automatically changing the screen orientation from portrait to landscape and vice versa which is particularly useful on small-screen devices as smartphones or PDAs as shown by [180] and implemented in recent mobile devices, like Apple's iPhone [6]. The later makes further use of orientation-sensing to implement specific user interfaces as demonstrated in a racing game [72].

All of the previously mentioned interaction techniques are possible by just capturing inertial orientation changes. However, tasks that a user wants to perform are often context sensitive. Therefore, combining orientation information with other context aspects to control the mapping will increase the usefulness of orientation-aware user interfaces. A mapping can thus be controlled by, e.g., the application status/operation profile, user interaction, device orientation, status of the environment (e.g. lighting), or location of the user. Much work in this area contributes to the topic of location-aware or, in general, context-aware systems [12, 35, 180, 66]. A generic framework integration of an orientation-aware user interface—implementing some of the aforementioned concepts for mapping orientation to user interaction—and an evaluation based on various prototypes is discussed in Section 4.5.

2.2.4 Exploiting Optical Character Recognition

Primary sensors to capture context data are image sensors, i.e. cameras. They are able to acquire huge amounts of data within short time intervals but require a subsequent processing—which is

[2]Marble Madness by Atari Games: http://en.wikipedia.org/wiki/Marble_Madness

often computationally very expensive—in order to gain knowledge from the data. However, most mobile devices already embed imaging sensors. Therefore, computer-vision based algorithms to gather context information are popular despite their computational load.

For the basic context attribute *position* mobile devices incorporate additional sensors, most important a receiver for the global positioning system (GPS). Position information is a prerequisite to implement navigation software; however, without additional hardware installations GPS positioning cannot be used for indoor scenarios. Other systems for location tracking in indoor scenarios are available, but these often require an expensive installation of infrastructure hardware or do not provide sufficient accuracy for most applications, as detailed in Section 2.2.1. Therefore an alternative technique to acquire precise indoor position and orientation information with minimal requirements on communication network and CPU has been developed. First results of the proposed technique are presented in [55]. An application where this technique was applied and further improved is presented in cooperation with D. Lucke, E. Westkämper, and O. Siemoneit [132].

Location and pose estimation using computer-vision techniques has been intensively researched and sophisticated algorithms are available that provide impressive results. D.G. Lowe shows how scale and rotation invariant image-space features (SIFT) are used for object recognition [131]. However, extensive computational load of these techniques make most of them inappropriate for small mobile devices. Instead of finding adequate natural features, marker-based approaches identify objects based on preinstalled markers. H. Kato and M. Billinghurst [109, 110] invented the AR-ToolKit which is a well known implementation of a real-time marker-based positioning system that is based on optical markers. Similarly, the Semapedia project [171] makes use of 2D barcodes to provide hyperlinks of real-world physical objects with an attached code to internet web pages.

Although the reliability and performance of marker-based approaches is high, the disadvantages of markers that have to be deployed and are attached to objects strongly limit the usefulness of these approaches. In contrast, text phrases offer features that are easy to recognize and often already installed at visible locations in the environment; labels, signs, door plates, posters, etc. are the most common examples. A tradeoff between generic object recognition based on natural features and recognition based on artificial markers is presented in [55, 132]: Existing unique text phrases of the environment—placards, posters, door plates, etc.—are captured with the mobile device's camera and analyzed using optical character recognition (OCR). The resulting phrases are looked up in a pre-acquired index that maps of text passages to locations. OCR is a well studied problem and can be solved even on mobile devices as shown by Koga et al. [118] and L. Jagannathan and C.V. Jawahar [102].

The main advantages of using OCR and a mapping database to implement a location system compared to existing systems are therefore:

- No hardware installations are required
- Utilizes existing low-cost hand-held devices
- Highly accurate position and direction information
- Only moderate computation capabilities are required

- Privacy-aware infrastructure-independent localization

Location-Determination Process

The proposed localization system is separated into three components: A client application to capture images containing text phrases, an optical character recognition service, and a database to perform text matching. Thereby, novel applications are supported due to the now available additional context data, like support for indoor navigation tasks or access to additional information of real-world objects. An overview of the system is depicted in Figure 2.9.

The estimation of location and orientation is implemented in a semi-automatic approach: The mobile users initiates a location request by taking an image of a real-world text passage. Afterwards, the OCR service is executed on the acquired image by the client or the server, dependent on the client hardware capabilities. Thereby, the preferred configuration is an execution on the client side since then the data amount transmitted to the server is minimized, i.e. only a character string vs. an entire camera image.

The resulting text is then transmitted to a database server for a search of matching text strings in the spatial database. This database stores context-enriched texts, i.e. arbitrary metadata including position, orientation, and translation that reference text passages. These metadata are sent back to the client application allowing sophisticated applications to be executed on mobile devices including, e.g., navigation or text augmentation/translation.

A similar system is proposed in [132] as shown in Figure 2.10. In contrast to the previous configuration, the enhanced system supports a server-independent operation. This way, the location of the user is protected from the infrastructure and therefore maintains the user's privacy, i.e. users cannot be tracked by the system. Without a communication to the server, the client device is required to perform additional tasks: hosting of the mapping database and string matching of the recognized text phrases. An entire or partial transfer of the database to the client is possible, e.g. when entering a building, since the data amount is small (about 800 kB for 1000 locations including metadata). However, databases with many text phrase entries will slow down the matching process on limited devices so that a partitioning of the entire dataset might become necessary, e.g. one database per floor of a building.

Requirements for OCR-based Positioning

OCR algorithms are very sensitive to the size of characters in terms of pixels, i.e. images where characters are represented with very few pixels cannot be processed reasonably, although the text is readable by humans. Several experiments with different resolutions, varying text sizes, and changed view distances have shown that cameras of mobile devices should at least support VGA resolution (640x480 pixels) for the proposed system. Higher resolutions will improve the OCR results but also increase the required processing power and—depending on the setup—the amount of data transmitted to the server. Current mobile devices like PDAs or mobile phones embed cameras with min. 2 MP (1600x1200 pixels); at the same time the network bandwidth varies from Mbit/s (IEEE 802.11 WLAN or Bluetooth) to few kbit/s (UMTS or GPRS). Therefore, VGA-resolution images provide a good tradeoff.

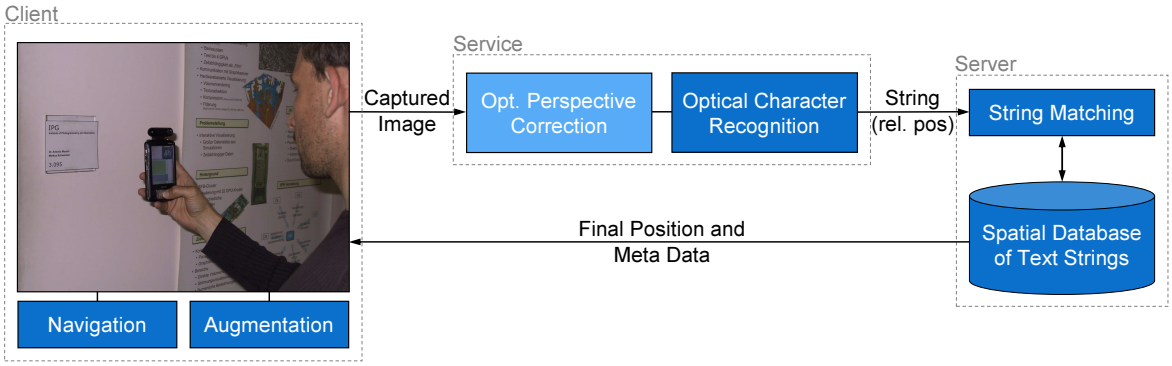


Figure 2.9: Overview of the optical-character-recognition-based location system using a client-server implementation.

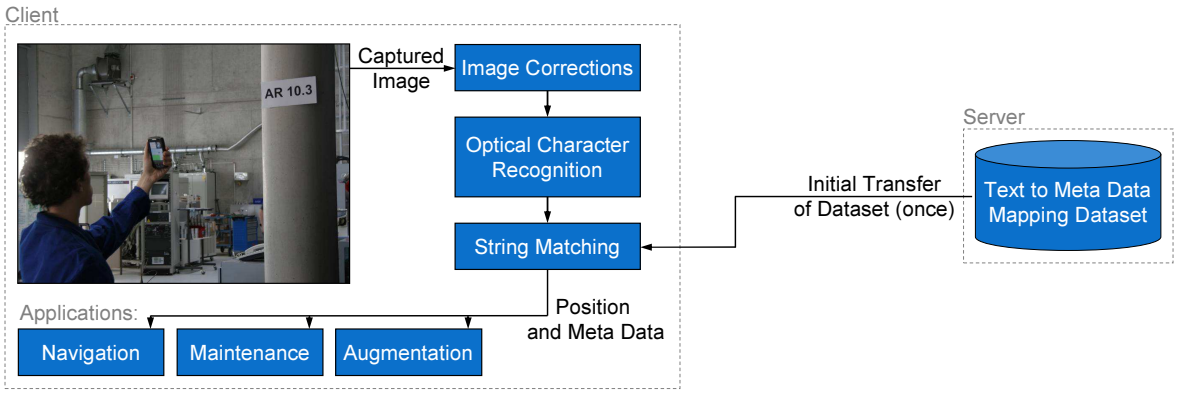


Figure 2.10: Realization of a privacy-maintaining location system. The user's location is evaluated on-demand by the client device without communication to infrastructure components and therefore protected against misuse.

In order to be able to deploy the proposed system in an environment, unique text phrases have to be identified. Good examples for such text phrases inside buildings are: Door plates or room numbers, unique placards or posters, labeled access ports or switches, etc. It is also possible to support non-unique text phrases, however, the string matching then provides several possible locations and the client application is responsible to handle this uncertainty (see Section 6.2.1). The text passages, in the following referred to as text entities, have to be pre-acquired and stored on a per-line basis in a database and linked to a text entity with possibly additional information like, e.g., the position of the text inside the building. During the evaluation of the proposed prototype system we noticed that the low-cost cameras of mobile devices often suffer from extensive lens distortion; this hinders reliable character recognition. Therefore, we propose to optionally let users calibrate their client camera once before usage, e.g. using techniques as available in ARToolKit [110]. A predefined pattern has to be captured from multiple view points; based on these the software can generate parameters that describe the camera's lens distortion. This information is later used to correct camera images for an improved OCR quality. If a client device does

not offer enough computation power, raw camera images can be transmitted to a server-based service (see Figure 2.9) including client specific camera calibration data (<100 byte) within the location query so that the server can perform the correction of the image.

Text Recognition and Text Matching

The processing starts with an optional distortion correction of the captured image. Again, dependent on the OCR service implementation the processing takes place on the client or server side using camera calibration parameters provided by the client. Afterwards, techniques to estimate the perspective distortion of the text within the image are executed. This is also an optional step, but without perspective correction users are forced to capture the images in an orthogonal view, i.e. the captured images have only minimal perspective distortion. Limiting our system to text strings, techniques for perspective correction based on the text layout are applicable, as presented by S. Ferreira et al. [71] or by L. Jagannathan and C.V. Jawahar [103]. Still, perspective corrections on images that suffer from extreme perspective distortion—more than ∼45 degrees off from an orthogonal view to the text—often result in inadequate quality for OCR processing due to the limited VGA resolution; images with higher resolution can help to alleviate this problem.

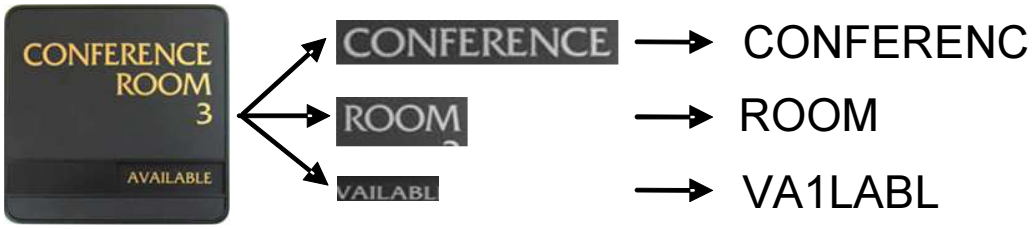


Figure 2.11: Steps of the optical character recognition: color reduction, segmentation (left, middle), and optical character recognition (right).

The perspective corrected images are then used for the actual optical character recognition. First, the text layout of the images is analyzed, i.e. where are lines of text, single column vs. two columns format, etc. Afterwards, the segmented image parts are separately sent to the OCR system, as shown in Figure 2.11. The resulting text lines are then compared to all text strings stored in the database using a fuzzy comparison algorithm with respect to OCR failures. For the string matching a greedy recursive maximum substring matching algorithm was used as proposed by M.D. Garris et al. in [78]. In addition, the word-error scoring was adapted to handle character-matching errors that possibly originate from OCR failures differently, i.e. weight them lower. The modified rating function uses a mapping table of recognized characters to a set of possible original source characters (see Table 2.1) and checks if matching errors disappear if the recognized character is replaced by one of the possible source characters.

This way the recognition result depicted in Figure 2.11 for the word "AVAILABLE" fails to match the "1" character, but as "I" is within the set of possible source characters it is not ranked as a full character-match failure. As a result of the string comparison the shortest distance (i.e. smallest error) to a string of the database and its ID are stored. The results of recognized lines which belong to the same text entity are combined to find the text entity with the smallest overall

Table 2.1: Character mapping of recognized character to possible source characters.

Recognized Character	Possible Original Source Characters
l	I, i, l, /, 1
O	O,0,c,o
...	...

error. The information associated with this entity—e.g. the entity's position and orientation within the building—and the quality of the recognition are returned to the client application, if the overall error is smaller than a configured error threshold. Otherwise, an error message that no text could be recognized is reported to the user.

Evaluation

For evaluation a prototype was implemented based on the open-source project OCRopus for OCR tasks [26] and an OCR-fault-tolerant string matching based on a method by M.D. Garris et al. [78]. During runtime the text-string database is entirely loaded into main memory, however the required memory of the service is still less than 4 MB for 500 text entities. For testing, databases of 500, 2000, and 5000 text entities—each 4 lines of text with about 4 words on average—were measured on a single core 2.8 GHz desktop machine.

The observed OCR performance depends on the captured image quality; noisy images noticeably slow down the recognition process. However, for 640x480 resolution images with moderate quality OCR processing takes less than 0.4 s. Naturally, string matching depends on the number of recognized words and on the number of text entities contained in the database. For the measured text scenario the performance was: 0.13 s for 500 entities, 0.59 s for 2000 entities, and 1.44 s for 5000 entities.

With the resulting fast, low-cost, and high-precision positioning system already many location-based services can be realized. Furthermore, the additional context information of high-precision orientation information—which is otherwise only available via expensive external tracking systems (Section 2.2.1)—allows even richer applications. Results of various prototypes that make use of this location system to implement location- and orientation-aware applications are presented in Section 6.2.1.

The drawback of this approach is that users have to guess adequate motifs and manually adjust/arrange the image sensor embedded in a mobile device, an automatic movement is virtually impossible. Applications are therefore required to either use a different, more appropriate sensor—if at all available—or cope with the fact that the sensor captured undesired data. To alleviate this discrepancy, a further option is proposed in cooperation with H. Sanftmann [58] to interactively guide the context-acquisition process.

2.3 Interactive Context Refinement

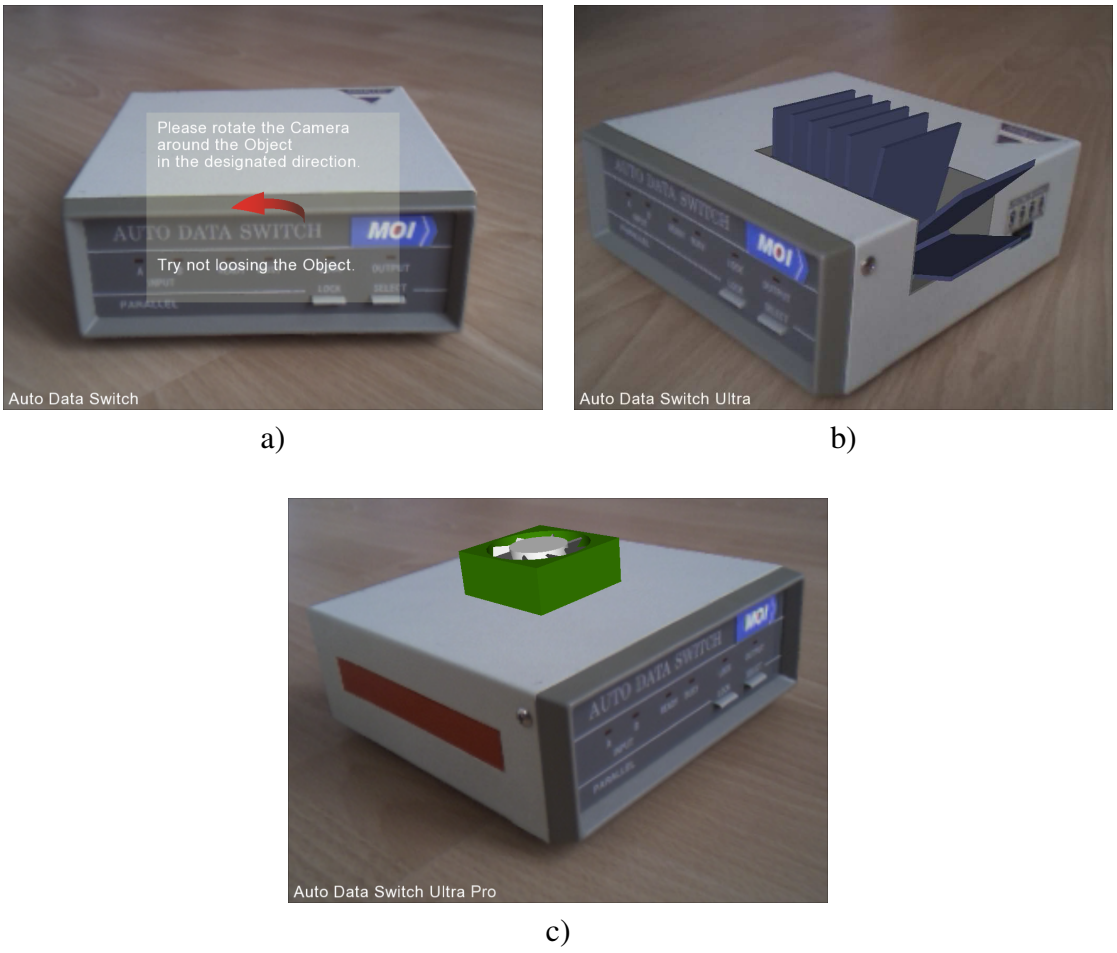
For maximizing the amount and quality of captured context information often a combination of multiple algorithms and sensors is used. Thereby, the combination of multiple data sources is a non-trivial task: Values may be in conflict or exhibit different accuracy. An improved technique to combine multiple approaches was developed in cooperation with H. Sanftmann and presented in [58]. The system shows a real-time interactive method to efficiently combine multiple techniques by considering the quality and significance of detected attributes. This is achieved by utilizing a hierarchical organization of specific context configurations that have to be detected. Furthermore, detection techniques are chosen based on context information—e.g. feature type, reliability of sensor data, etc.

As an example application the proposed context-refinement technique is used to implement an object-recognition and identification system that is able to efficiently differentiate a huge number of only marginal different objects, primarily utilizing computer vision algorithms. In general, distinguishing features might not be captured by any of the available sensors; only if specific sensor configurations are applied—e.g. view port, exposure, etc.—distinguishing features are detectable. The system is targeted for interactive augmented reality applications (see Section 3.4) where virtual geometry is accurately aligned with a real-world image as depicted in Figure 2.12b, c. Therefore, a primary challenge—when supporting multiple, possibly alternative, algorithms for object recognition—is to prevent a degradation of performance and quality due to numerous combinations of recognition algorithms and reference objects that have to be evaluated in realtime.

2.3.1 Hierarchical Decisions for Context Evaluation

In the research areas of artificial intelligence and computer vision decision trees are a common approach to divide and solve problems [173]. Although the following examples and the final prototype focus on object recognition, the basic approach is generally applicable for context evaluation. The advantage of using hierarchies for object recognition is proposed, e.g., by R. Mehrotra et al. [136]; they group distinctive features of objects to generate a tree and traverse it during the object recognition phase. Our work extends this general concept with support for multiple different techniques to evaluate decisions and the possibility to return intermediate results, detailed in following sections. Also, P. Viola and M.J. Jones [198] propose to use a degenerated decision tree to achieve fast recognition results. They concatenate multiple weak continue/reject classifiers to build stronger classifiers; however, nearly identical objects that cannot be differentiated in arbitrarily captured views are not handled adequately. M. Grabner et al. [82] use SIFT-like features to distinguish objects; the system groups similar objects in a hierarchical structure, but only a single recognition technique is utilized.

The proposed technique also makes use of a hierarchy to prevent performance degradation due to numerous algorithm-object evaluations. In addition, static and dynamic context information is utilized to traverse this hierarchy, whereby ambiguous decisions are solved by requesting (manual) sensor adjustments.



a) b)

c)

Figure 2.12: Steps of an interactively refining object recognition: a) recognition of the object class *Auto Data Switch* and presentation of a hint to capture further distinguishing features, as seen in b) or c). For illustration, different models of the *Auto Data Switch* are augmented with enhancements as shown in b) and c).

Approaches that specifically address the problem to find and configure an optimal sensor adjustment—most often a camera-view adjustment for computer vision based approaches—are termed as *active vision* systems. These techniques build rules to, e.g., move the camera to an improved view, which allows further refinement of the object recognition. However, most active vision approaches assume that an automated camera movement is available, e.g. a robot arm, which is practically impossible for hand-held devices. H. Borotschnig et al. [23] present a comparison of three approaches for these so-called active object recognition systems. The examined techniques are based on different uncertainty calculi: probability theory, possibility theory, and the Dempster-Shafer theory of evidence. M. Reinhold et al. [165] present a statistical appearance-based object recognition approach combined with an active view point selection.

Conceptual Overview of the System

Interactive context refinement provides a piecewise refinement of available context information to build an enhanced context. In the example of an object-recognition application, context information—knowledge about the object to-be-recognized—is bit by bit enhanced: First, a rough object shape is detected and the orientation is estimated so that context information now contains a object shape and orientation. Using shape and orientation, subsequent context aspects are evaluated, e.g., check for a blue color spot on the object's right side at a specific position and add the result to the context information. Such a recognition process is executed by traversing a hierarchy of distinguishing features; thereby the number of possible object matches is continuously reduced. Furthermore, external context information might be queried, if considered beneficial for the current recognition, e.g. available lighting condition to adjust camera parameters. The hierarchy also allows to present intermediate recognition results, e.g. object (sub-)classes, and trigger actions that are needed to further descent the hierarchy, if the system cannot differentiate an object (i.e. cannot continue traversing).

The architecture of the developed context refinement system is exemplarily shown for the application of object recognition tasks in Figure 2.13. Algorithms that detect or differentiate context aspects need some information from the knowledge database (also referred to as metadata store). In the application of object recognition, this can be reference images, texture information, 3D models, or any other information. This kind of information—in the following referred to as metadata—is typically generated in an off-line preprocessing task for each node and stored, as shown in Figure 2.13.

The data contained in the metadata store is primarily accessed by integrated context recognition algorithms (solvers). The framework is based on the concept that multiple different solvers—seen in the center of Figure 2.13—are utilized during the context-refinement process. This way, arbitrary techniques using various metadata can be integrated and combined to calculate enriched context information. In addition, solvers have access to a further data source the so-called *Context Store*.

The Context Store's content can be considered as data that describes the current conditions of the application, environment, or any other attribute which might dynamically influence the refinement process. This store is used for locally acquired sensor data, intermediate results, and also for *external* context data, e.g., provided by Nexus.

The previously mentioned hierarchy of distinguishing features is shown in the upper left corner of Figure 2.13. The traversing of this hierarchy is controlled by a simple controller which executes the referenced context recognition techniques (Fig. 2.13, dashed arrows).

2.3.2 Organization of Context-Evaluation Nodes

The hierarchical structure used for the refinement utilizes several node types that define different behavior triggered during the traversal of the graph. A basic node type is the *Search Node* F. It references multiple alternative solvers S_x that are adequate to iterate through all of the Search Node's children and search for the best matching. This behavior corresponds to an recognition approach without an underlying hierarchy, where all possible solution (i.e. objects) M_x are

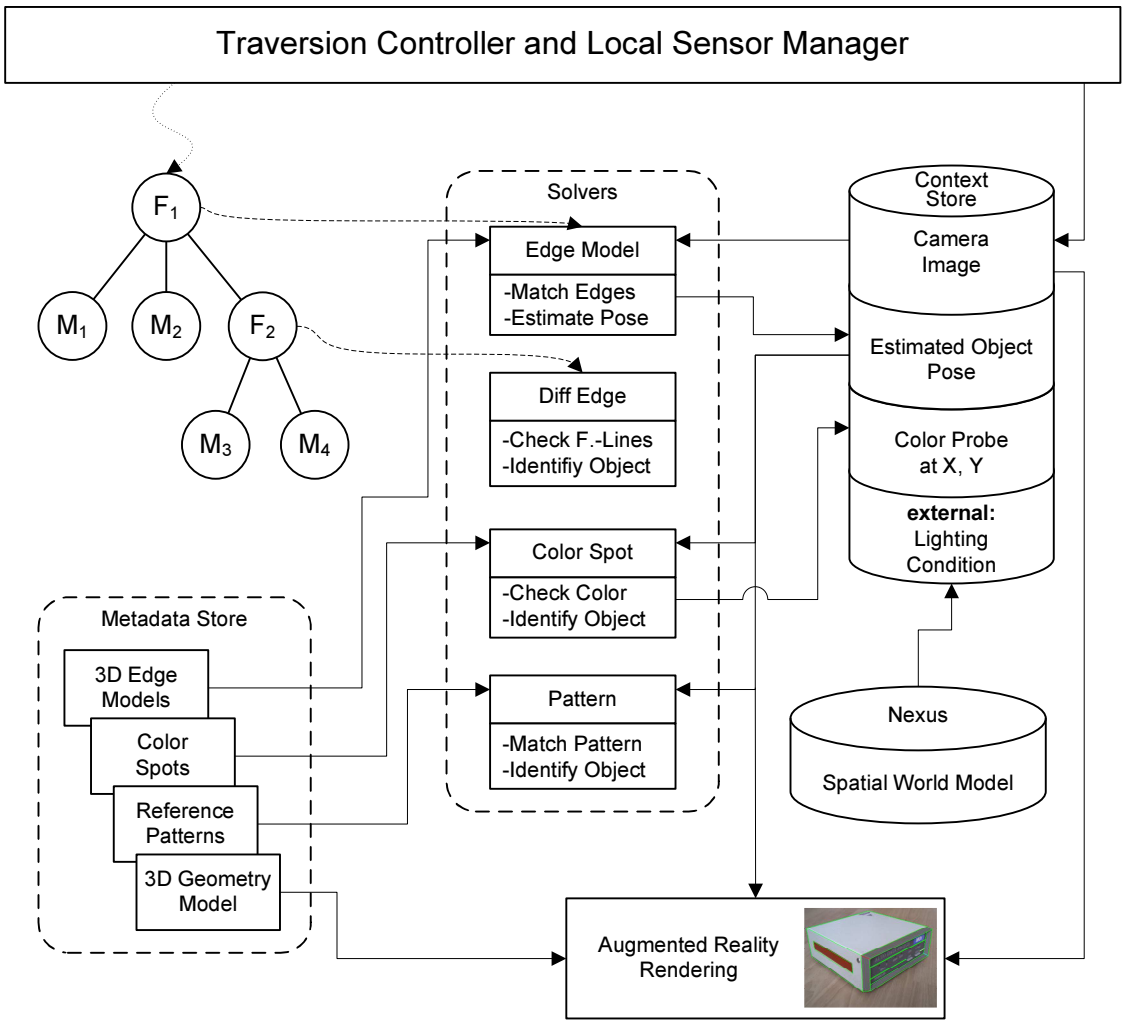


Figure 2.13: Overview of the iterative context refining system.

checked for a match in sequence; an example for this setup is shown in Figure 2.14. During traversal, solver S_A and S_B can access corresponding metadata, referenced by the current node, e.g. for the application of object recognition these could be distinguishing line features of real-world objects.

An advantage of the hierarchical structure is that it allows to group similar context configurations based on common context aspects beneath a group node. These intermediate group nodes reference common metadata of all their children that is stored in the *Metadata Store*. The framework is able to present intermediate results of the context refinement process even if it cannot entirely differentiate all context aspects, referenced by a Search Node. Applications may already benefit from such intermediate results, e.g., to show a coarse 3D proxy model presenting the common appearance of the entire model group (see Section 2.3.4). An example configuration is depicted in Figure 2.15. The Search Node F_1 will execute solver S_A during context-refinement traversal to differentiate models $M_{1..3}$ and the group of context aspects subsumed under F_2; $M_{4..6}$

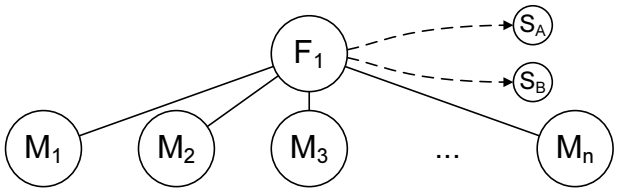


Figure 2.14: A simple refinement setup which linearly checks each candidate M_x, without the advantage of a hierarchical structure.

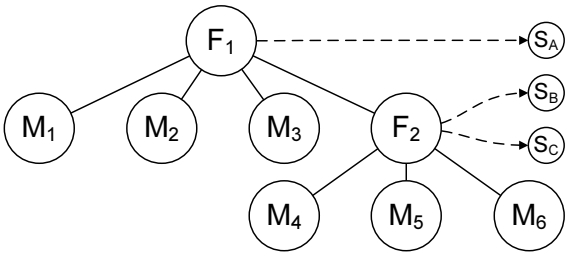


Figure 2.15: Hierarchical organization of distinguishing features. F_1 differentiates $M_{1..3}$ and the group F_2; $M_{4..6}$ are separated via F_2.

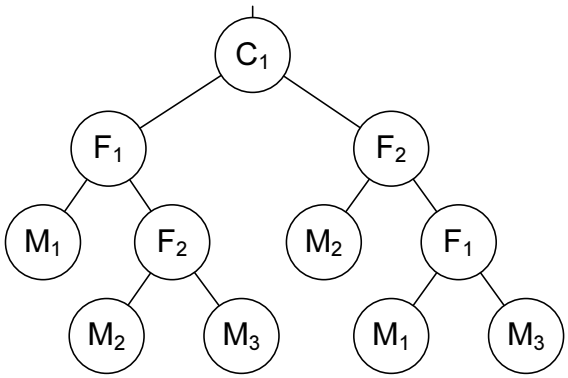


Figure 2.16: Alternative sequences for the recognition process via context-switch C_1.

are distinguished based on the referenced metadata of F_2.

Context-Switch Node C is a node type that is able to route the traversal of the tree dependent on context attributes. Context-switch nodes can also have a number of child nodes but do not reference any solvers; they only evaluate context data to decide how the traversal continues. This way, alternative sequences for the refinement can be integrated into the mostly static hierarchy. Figure 2.16 shows a configuration where the context-switch node C_1 decides into which child the traversal descends based on context information. As can be seen in Figure 2.16 the two sub-graphs are simply permuted versions of each other. This way, sequences for which the system performs best—e.g. without the need to request user interaction (Subsection 2.3.3)—can dynamically be selected.

2.3.3 Incremental Accumulation of Context Aspects

The incremental context refinement is performed by traversing the hierarchy and, dependent on the node type, processing the results after executing the referenced solvers to select the child node in which to descend. Exemplary setups of hierarchies are illustrated in Figure 2.17. Multiple solvers are integrated that identify context aspects of completely different real-world characteristics: 3D geometry, color, or pattern. A more detailed description of the utilized techniques for the prototype implementation is given in Subsection 2.3.4. An important aspect of the proposed system is that the knowledge gained from the current and previous iterations of the context

refinement, e.g. the object's orientation, is gathered and stored as context information. During traversal of the recognition tree, subsequent solvers have access to this context information and may consume, correct, or extend it with additional aspects.

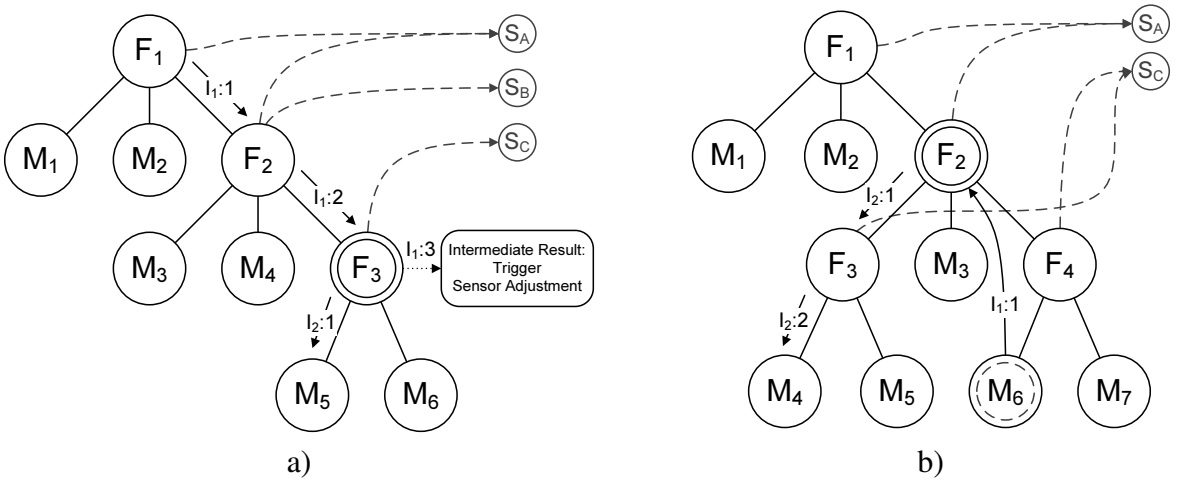


Figure 2.17: Hierarchical organization of knowledge data via multiple search nodes F_x. The refinement performed in a) stops at the intermediate model referenced by F_3. Afterwards a trigger is executed to request a sensor adjustment (I_1:3) to capture further distinguishing features. In b) previously identified context aspects have changed, and therefore the system performs a backtracking (I_1:1) and initiates the next iteration at F_2.

If a leaf of the hierarchy is reached during the traversal then all context aspects that are considered by the system have unambiguously been identified and the refinement task is finished, returning the gathered context information (see Figure 2.17a, following steps I_1:1, I_1:2, I_2:1). An important advantage of using a hierarchy for organization—in the object-recognition example nodes of the hierarchy group models with similar features—is that our system is able to descend into the hierarchy as soon as an adequate match is found in the referenced data, as similar aspects will be summarized using a group in the hierarchy. In contrast, simple approaches would have to linearly check *each* referenced occurrence for a match to find the best matching which could potentially be the last reference.

Whenever a node can distinguish its children no further, and therefore cannot further descend into the hierarchy, intermediate context results are returned. For that purpose, our system supports an interactive refinement of intermediate results by triggering actions that help the system to further differentiate the context as can be seen in Figures 2.18, 2.12a, and 2.17a, following steps I_1:1, I_1:2, and I_1:3. This feature can be used, e.g., to instruct the user to perform appropriate camera movements and optimize the point of view to capture additional features (see Fig. 2.17a step I_2:1). Within the implemented prototype actions were limited to camera-movement (Fig. 2.12a) and light-adjustment (Fig. 2.18) commands, but other actions—e.g. adjusting camera shutter/focus, manual user decisions, or activation of additional sensors—might easily be triggered.

For achieving a high performance in the refinement phase the hierarchy helps in two ways: First, the number of possible configurations that have to be searched by Search Nodes are reduced

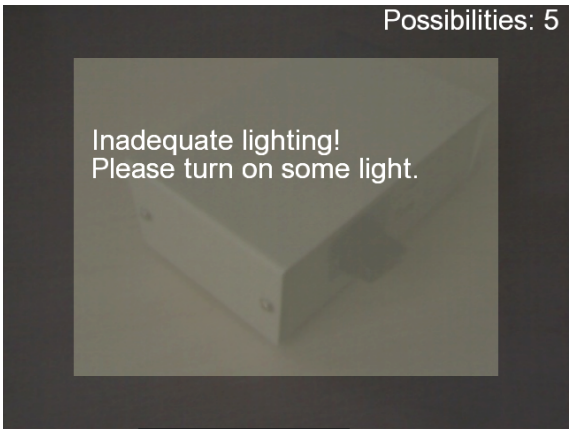


Figure 2.18: In situations where the system cannot further decent the hierarchy, requests are triggered to adjust, e.g., the lighting condition.

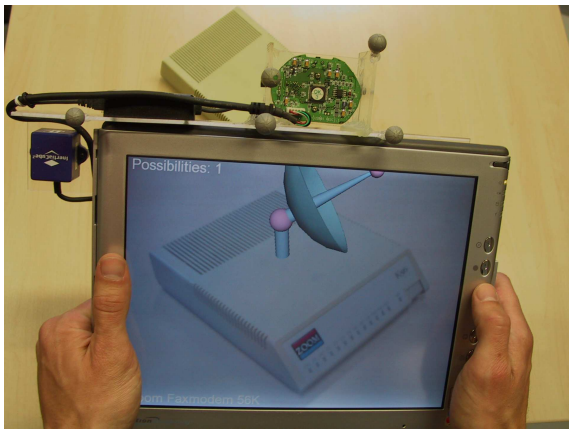


Figure 2.19: The prototype hardware in use. A TabletPC was equipped with a webcam (top middle) and an inertial sensor (top left).

due to the tree-like structure; second, the temporal coherence—i.e. in most subsequent iterations the same context aspects (i.e. the same object) are captured—can effectively be used to skip large parts of the hierarchy. This is achieved by starting the traversing at the node returned in the previous iteration, symbolized as double-framed nodes in Figure 2.17. If the starting node is not the root, i.e. when temporal coherence is utilized, the system has to check if the to-be-refined context is still the same. Furthermore, some context aspects might have got invalid since the previous frame and have to be updated, e.g. due to slight sensor movements. Therefore, the system has to ensure that the information required by subsequent nodes is up-to-date by executing the corresponding solvers. This dependency can be determined and stored in a preprocessing step. For the prototype implementation we optimized the restart by using a combined solver that checks for a change of the object and estimates its new pose. The system simply checks if the object is still the same by trying to match line features that are referenced by the node where the recognition process continued. The matching is executed quite fast since line features are already known and a good approximation of the camera position is provided as a result of the previous iteration. For the prototype (Section 2.3.4) the estimation of the camera movement between subsequent iterations was improved using an inertial sensor to measure the acceleration and approximate the new camera position. This way, further context information is integrated to improve the performance of the system.

If the matching returns a positive result, the context update is stored, and the algorithm continues traversing as described in the first paragraph and thereby all recognition results of previous iterations are kept. This is illustrated by step I_2:1 in Figure 2.17a. In contrast, if the verification fails to match enough features, the system assumes that the situation which has been recognized in the previous iteration has changed and therefore has to check if other context aspects of the hierarchy match the current scenario.

After detecting that the context has dramatically changed, the system can simply *reset* and

start the refinement from the root of the hierarchy. This, however, will result in an inefficient behavior whenever a set of context aspects, e.g. object pose and some line features, cannot be recognized reliably in subsequent iterations: The system will have to descend the entire hierarchy every time a solver is unable to detect enough context aspects. To overcome this limitation a simple backtracking mechanism is integrated to ensure that the system remains efficient, i.e. the refinement is restarted from a previous node on the path back to the root. This recursive process continues until it is able to descend the hierarchy again or—in the worst case—a restart of the refinement is initiated at the root node of the hierarchy.

During construction time of the knowledge hierarchy a link can be stored per node that is followed during backtracking to skip in-between nodes in order to increase the efficiency, as shown in Figure 2.17b step I_1:1. Afterwards, the following iteration (I_2:1, I_2:2) traverses again to a leaf node.

In the proposed interactive refining special cases occur that are annoying for users: The system might request a sensor adjustment, e.g. camera movement to the right, to continue the traversal. However, a following search node might request an opposite adjustment, i.e. a movement back to the left; whereby it possibly would have been satisfied with the setup at the beginning. In worse cases, users are required to adjust sensors, e.g. the camera view, multiple times whereas one adjustment would have been sufficient. Our system overcomes such scenarios by integrating Context-Switch nodes C to select alternative sequences for recognition. The sub-graphs of C nodes are simply permuted in their order of execution as illustrated in Figure 2.16 and selected based on the current context.

2.3.4 Prototypical Refinement Application and Results

For the evaluation of interactive context refinement a prototypical application was implemented. The software focuses on the challenge to differentiate real-world objects with only marginal distinguishing features that can often only be identified from specific points of view and solves this problem by interactively guiding the user during the recognition process. As illustrated in Figure 2.13, the prototype implements different object recognition techniques as solvers whereas corresponding data of the models are stored in the metadata store. The hierarchy is built based on distinguishing features of the to-be-recognized models. The object-recognition techniques are adapted and partially improved versions of well-known algorithms. In particular, an approach proposed by M. Dhome et al. [49] is used to find an analytical solution for the pose of a 3D object in space. Simplifications for the special cases of coplanar lines and three-line junctions are given which reduce the problem to fourth-degree equations. The performance of this method even allows an application on mobile phones as presented by D. Beier et al. [19]. D.G. Lowe [130] presents an algorithm that iteratively refines an initially guessed view point via Newton's method.

The presented recognition system has been specifically designed to share the context and metadata store with other parts of the application. Therefore, augmented reality visualizations (see Section 3.4) can easily access the position and orientation information using the context store to align presentations of virtual information with real-world images. The prototypical implementation of an interactive assistant system to help users to identify and augment objects makes use of this concept. An exemplary target application for the prototype is an information system for

customers and consultants who are interested in HIFI appliances. These items have many similar aspects which cannot easily be differentiated. Furthermore, augmented reality provides an intuitive way to present different instances of an object, which is probably not yet available, like extra attachments, different colors, or even custom case modifications.

Implementation of Solvers and Information Presentation

The most advanced solver is—in addition to object identification—also able to estimate the pose of objects. It is implemented using computer vision methods: First, feature lines and three-junctions are searched in the captured image. These are linked to the model's geometry description—stored in the metadata store—to generate hypotheses of possible models and their orientation [49]. The second part of the solver is based on a technique proposed by Lowe [130] and is used to check generated hypotheses and further improve the pose-estimation accuracy by minimizing the matching error. This solver is referred to as *edge-model solver* S_E (see Figure 2.13).

A cut-down version of the edge-model solver is integrated to differentiate similar models where one model has additional feature lines. A prerequisite for this solver is a pose-estimation solution, calculated by any previous node in the recognition tree. The solver can then project the additional line features and search for matching edges in the captured image (*diff-edge solver* S_D). Distinguishing features, e.g. additional lines, might be visible only from a specific point(s) of view therefore a differentiation of the objects based on a captured image from an arbitrary view is not always possible and an intermediate result might have to be returned. The novel approach of the proposed interactive refining technique to solve such undetermined cases is that solvers can report intermediate results with hints/triggers that state which changes have to be made in order to continue the recognition. This includes requests for changing, e.g., camera position or lighting conditions, which are displayed to the users as can be seen in Figure 2.12a and Figure 2.18.

The *color-spot solver* S_C is able to check the color value at predefined locations on the object surface. It simply projects relevant color probe locations using the previously estimated object pose to image space and examines the pixel color in the captured image. Further, similar real-world objects are often labeled or marked in order to express their differences. The prototype therefore supports an approach to compare previously acquired reference images to the captured image information to support an identification based on patterns. This *pattern solver* S_P benefits of a previously calculated pose estimation in two ways: It is able to determine whether the pattern is entirely visible in the camera image, i.e. it is not hidden or occluded, and the perspective distortion is a priori known due to the previously calculated object orientation and pattern location, stored as metadata. Many additional techniques can easily be integrated to detect more complicated features like a solver to recognize curved surfaces or flexible object parts. But also identification components like optical character recognition or barcode scanners can be applied.

For presentation the prototype implements a simple augmented reality rendering module that is able to present a live video image of the attached camera with an information overlay. If the system is unable to totally identify an object, only its category is displayed. As mentioned in Section 2.3.3, intermediate nodes may trigger actions to improve the recognition, e.g. a trigger

to initiate camera movement requests may display messages to guide users interactively, as seen in Fig. 2.12a and Fig. 2.18. For the proposed scenario, presentations based on augmented reality further benefit from the possibility to show different virtual instances of objects, similar to the illustrations in Figures 2.12 b, c. These renderings depict two different augmentations, which might represent future configurations or not-in-stock items.

Evaluation and Results

For the evaluation of the prototype implementation a dataset with five computer-appliance objects were used. Two objects are identical except for a red stripe on one object. Therefore, these objects can only be distinguished if the red stripe is within the camera's view. A direct comparison to existing systems cannot be given since the proposed setup and target application is rarely examined, often a specialized algorithm is utilized where the objects used to evaluate the system fit to the proposed algorithm. The theoretical complexity in terms of executions of a *solver-model* pair in the hierarchical implementation is optimally $O(log(n))$; n denotes the number of different objects to-be-recognized. For a simple linear search method the complexity is $O(n)$ which corresponds to the worst case of our approach. In practical setups the system therefore achieves a performance in-between both extremes. However, these numbers strongly depend on the number, type, and quality of the objects and the structure used for the recognition tree. The implemented recognition algorithms and the selected techniques also have a great influence on the overall performance.

Measurements in Table 2.2 present the performance for an Intel Core2 Quad Q6600 CPU running at 2.4 GHz (only a single core was utilized) for both cases: without and with utilization of previous recognition results. The video stream of the camera was simulated with a static 320x240 resolution image in order to achieve comparable results.

Table 2.2: Performance comparison for the non-coherent (re)start of an object-recognition process and a refinement operation utilizing temporal coherence.

	non-coherent	*coherent*
preparation	21.3	21.3
hypOther	163.1	-
hypValid	5.8	-
hypCheck	0.8	0.8
rendering	1.4	1.4
Overall	192.4 ms	23.5 ms

The first phase is identical in both cases: Undistortion of the camera image, generation of a monochromatic image, execution of Sobel/Canny edge detection, and the merging of collinear line fragments which is summarized as *preparation*. In the *non-coherent* case, where no object registration and pose estimation is available from previous frames, a large number of hypotheses have to be evaluated. The timing values of *hypOther* refer to hypotheses that are evaluated

with 3D object models that do not correspond to the captured camera image. The *hypValid* measurements refer to hypotheses that are evaluated with a matching 3D object model. In the *coherent* case a valid object pose is already available from a previous iteration, as the camera is fixed during the evaluation. Timings of *hypCheck* refer to the Lowe-based approach for checking and improving the pose-estimation hypothesis. Timings for display of the captured camera image and optional augmentations are summarized in *rendering*.

The measurements show the benefit of utilizing temporal coherency as the most time-consuming part of the algorithm is efficiently skipped. With the previously mentioned performance improvement due to the utilized hierarchy the system is especially suited for interactive real-time applications.

Overall, the interactive refinement of context information enables a selective enrichment of the known context by interactively guiding the recognition process. Triggered sensor adjustments—either for automatic or for manual reconfigurations—are effectively used to detect additional relevant context aspects. The system further allows combining arbitrary recognition techniques using a hierarchical structure to maintain high efficiency.

3 Computer Graphics and Visualization

The term *computer graphics* roughly summarizes everything related to generating images or illustrations using computers. Computer graphics evolved from simple line-based illustrations to 2D images up to nowadays popular 3D graphics. The process of image generation, also termed rendering, is either realized using software algorithms or via dedicated graphics hardware. Software implementations offer more flexibility and are therefore often applied for high-quality results; however, hardware-based approaches are (in general) faster, allow interactive feedback, and real-time graphics. Modern hardware graphics accelerators include a graphics processing unit (GPU) that implements the abstract concept of a rendering pipeline based on rasterization.

One of the most important tasks computer graphics can be use for is the presentation of information. Thereby, possibly huge amounts of data is processed, transformed, and rendered according to the visualization pipeline, which is also extensively used in this thesis. Especially interactive visualization techniques are important to support users in understanding data and behavior of various phenomena.

Artificial, computer generated visualization can further be integrated into a real-world context (mixed/augmented reality) or can even be used to completely replace the physical environment (virtual reality). This enables data visualization to be presented within its relevant context, physical or virtual.

3.1 The Rendering Pipeline

This pipeline was designed for 3D graphics and provides a theoretical model of steps that are performed for each primitive that contributes to an image. Currently, hardware accelerated graphics are implemented using the rasterization paradigm, i.e. primitives (triangles) are projected to the 2D output domain and are filled (*rasterized*) to generate pixels. Figure 3.1 shows the conceptual model of the rendering pipeline, implemented by modern GPUs. Note that the actual physical implementation a GPU might differ, but existing APIs (DirectX or OpenGL) are designed based on the rendering pipeline concept.

In addition to many parallel processing units—in order to perform many tasks of the rendering pipeline in parallel—also modern GPUs embed fast, local memory. An Nvidia GeForce 280GTX, e.g., integrates 240 processing cores and offers a memory bandwidth of 141.7 GB/s.

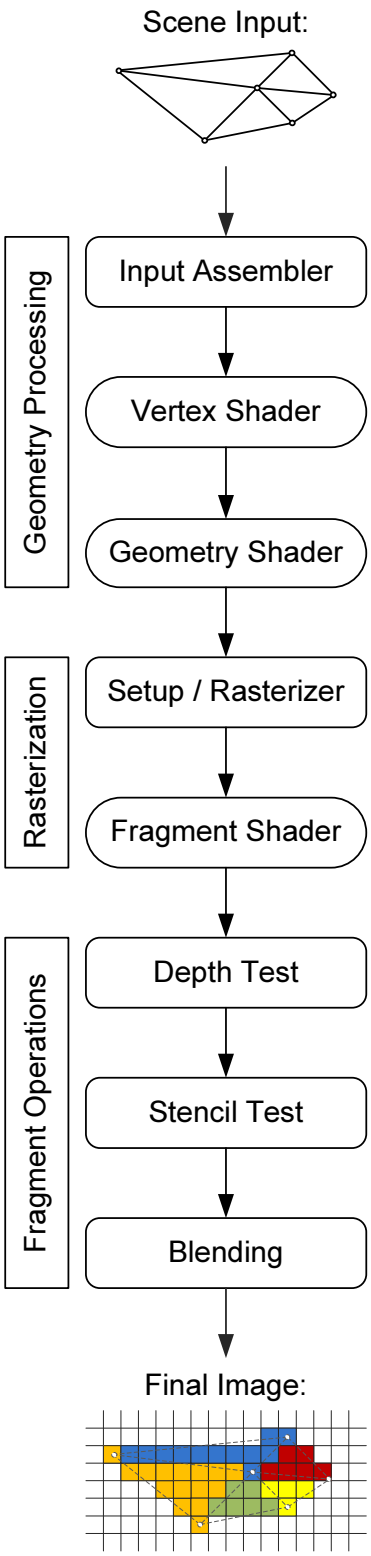


Figure 3.1: Rendering pipeline.

The pipeline can logically be divided into three steps: geometry processing which operates on the 3D input geometry, rasterization responsible for generating fragments (pixels with additional information), and fragment operations that convert fragments to pixels. On modern GPUs several stages of the rendering pipeline are fully programmable: vertex shader, geometry shader, and fragment shader, shown with circular frames in Figure 3.1. Most other stages are currently *configurable* but will most likely be programmable in future GPUs.

Geometry Processing starts with the input assembler that is responsible to supply data to the pipeline, possibly queried from the GPU's local memory (triangles, lines, points, vertices, texture coordinates, etc.). Afterwards, each vertex of the input is piped through the vertex shader which generates per-vertex output data based on the currently active vertex program. Typical operations performed by a vertex program are: (model, view, perspective) transformation, usually in homogenous coordinates via 4x4 matrices, skinning, and lighting. Subsequently, vertices are assembled to primitives that are transmitted to the geometry shader stage. Per-primitive operations can be performed using a geometry shader program like generating additional vertices or discarding primitives.

Rasterization generates fragments independently for each primitive received from the geometry shader. Therefore, primitives are first clipped to the virtual camera's view frustum, still in homogenous coordinates. Afterwards, the primitives are rasterized (filled), i.e. the values of fragments covered by the primitive are calculated, and thereby the homogenous coordinates are projected to screen coordinates. The actual computation of a fragment's result is performed by triggering the execution of a fragment program for each fragment of a primitive. Typical input data for fragment programs are textures (bitmap images) and per-vertex attributes—like texture coordinates, normals, or material properties—that vary across the primitive and are calculated by the rasterizer via interpolation.

Fragment Operations define several tests and combining operations that each fragment has to pass in order to be written into the result image. Most important the depth test which ensures that only nearest fragments are visible, hiding fragments of other primitives further behind in a 3D scene. The stencil test works together with a stencil buffer and is used to perform per-fragment counting and clipping tasks. Before a fragment is written to the resulting image as a pixel, it is optionally blended with the previous content of the result image. This way, resulting fragment colors can be added or multiplied to achieve effects like transparency. Usually the resulting image is displayed on a output device, but it can also be interpreted as a texture and reused as input to one of the programmable stages. Even multiple such dynamic textures can be generated concurrently using multiple render targets (MRT).

3.1.1 GPU Performance

When rendering complex scenes using graphics hardware, many aspects of the rendering pipeline are executed in parallel, most important the programmable shader stages. Motivated by this intrinsic parallelism GPUs are designed as SIMD (single instruction multiple data) processors and thus reach a peak performance of 933 GFLOPS/s (Nvidia GeForce 280GTX) vs. 38.4 GFLOPS/s of modern CPUs (Intel Core2 Quad Q6600).

Besides the massive parallel design of GPUs, the high memory bandwidth and dedicated hardware logic is responsible for this high performance. But still most complex shader programs are in fact performance limited due to memory accesses like texture lookups. Peak performance is therefore seldom achieved, although GPUs include mechanisms to hide latency caused by memory accesses; thus, performance measurements are required in order to analyze and compare various GPU-based rendering approaches. In interactive applications this is often done in frames per second (FPS), i.e. number of full frame redraws that are achieved within one second, or by measuring the time a certain (very high) workload takes until it is finished. Definition of the term *interactive performance* varies and cannot easily be expressed in fps, but about 3-15 fps seems to satisfy most conventions.

Together with J. Diepstraten many different GPUs were tested to analyze their performance on various shader program instructions; the results are presented in [51, 54]. An interesting outcome is the performance for texture lookups, i.e. a memory read within a programmable shader stage. In general, a lookup addresses a specific location within a texture from where the data is read; additional filters (nearest neighbor/point filtering, (bi)linear, etc.) can be applied to interpolate between neighboring data values. Depending on the filter, more or less data have to be read: Nearest neighbor interpolation just requires the data value itself; whereas bilinear filtering interpolates between four data values. Data dependency is a further factor that influences the texture performance. Thus, so-called dependent texture lookups require calculations of the dynamic lookup location to be finished before the memory read is performed. Among others, these aspects are measured in [51, 54]; the result for texture performance is shown in Figure 3.2.

The measurement shows the duration for rendering 1500 quadrilaterals of 512x512 pixels in size. Different GPUs (color) and different texture types—four component textures with 8, 16, and 32 bit precision—were tested. Interestingly, for non-dependent lookups point filtering and linear filtering achieve equal performance, although linear filtering requires accessing a larger

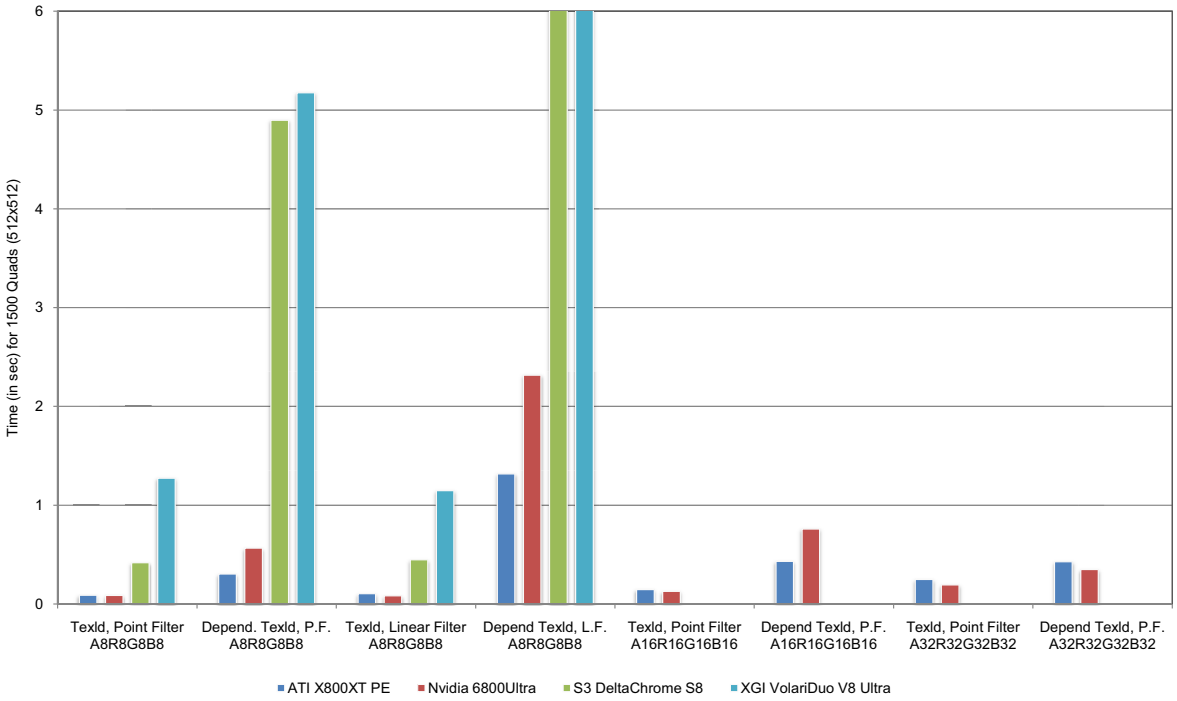


Figure 3.2: Texture look-up throughput on diverse GPUs.

amount of memory. This indicates that GPUs are optimized for specific use cases to achieve maximum performance when used in the most common applications for GPUs: 3D computer games. Thus, in praxis GPUs provide a very fast (bi)linear data interpolation, often at the same performance as single data value reads.

3.1.2 Application Programming Interfaces for Graphics

For programming GPUs two low-level APIs exist: OpenGL [183] and DirectX [139]. Both C++/C-interfaces offer similar functionality to configure and program the stages of the rendering pipeline in order to achieve the desired visual output, e.g. a photo-realistic water surface rendering as presented together with T. Klein and D.Weiskopf in [116]. While DirectX offers a fixed feature set and is therefore GPU independent, OpenGL allows vendor specific functions that are often only available on specific hardware. In contrast, OpenGL is platform independent and available for Windows, Linux, MacOS, etc.

Probably this platform independency led to the popularity of OpenGL on embedded systems. OpenGL ES 1.1 is based on OpenGL 1.5, therefore the embedded version 1.1 of OpenGL does not support programmable shaders. With the introduction of OpenGL ES 2.0, which is based on OpenGL 2.0, programmable shaders are available and required since fixed function transformations—i.e. the fixed default transformation and lighting computations—are no more available.

In contrast, for non-graphics related computation tasks performed on GPUs, to utilized the

huge computational power, other novel APIs have emerged: Nvidia's CUDA [148] and ATI's Stream [3] are proprietary solutions offered by GPU vendors; OpenCL [112] and DirectX Compute Shaders [139] are standards that will be supported by all GPU vendors. Most APIs are separated from existing graphic APIs but offer some interoperability to exchange data. In contrast, DirectX compute shaders are fully integrated within the DirectX API and therefore offer the most flexible API for combined computation and visualization tasks.

3.1.3 High-Level Shading Languages

Several stages of the rendering pipeline are fully programmable, similar to common CPU programs. However, GPUs have specific functionality and a different design compared to CPUs. Formerly, assembler language was required to program GPU stages, but now both APIs, OpenGL and DirectX, offer a C-like programming language with some extensions to address GPU-specific functionality. The syntax of the OpenGL Shading Language (GLSL) is similar to DirectX's High Level Shading Language (HLSL) and both offer basically the same possibilities.

In addition, DirectX provides the concept of effect files (.fx) to encapsulate most of graphics programming code into separate files. This way, vertex, geometry, fragment, and compute shaders including their class, structure, or variable definitions and configuration options for other stages of the rendering pipeline can be organized in separate files to increase maintainability of software projects. Also reusability of recurring functions is improved by effect files, as file containing shader code can simply be included within effect files. This is useful for, e.g., post processing techniques, which advanced visualizations often make use of to enhance the image quality or achieve certain effects (blurring, sharpening, etc.); some setups even calculate the entire lighting as post processing (deferred shading [42]).

Deferred shading is often implemented using a G-Buffer approach as proposed by T. Saito and T. Takahashi [174]. An extension to common DirectX effect files is presented together with D. Weiskopf to provide a generic implementation of GPU G-Buffers, also programmed using effect files [62]. The proposed framework makes use of textures that are updated with a desired G-Buffer attribute (intensity, color, normals, position, material ID, etc.) when rendering the scene's geometry. In a second step, these texture buffers are used to perform post processing techniques using only the information contained in the G-Buffer. Figure 3.3 shows an example: The geometry of the David statue is rendered to a texture storing intensity information. Afterwards, edge detection and dilatation is used to generate the final result.

3.1.4 Graphics Hardware support on Mobile Devices

Graphics capabilities of modern smart mobile devices rapidly increase, some even embed a GPU, but still most hand-held devices offer only limited graphics performance. Apart from that, interactive visualization applications require CPU resources for application logic, GPU task preparation and submission, or even for calculating the entire visualization in software. Unfortunately, most mobile devices also suffer from limited CPU performance, especially due to the absence of dedicated floating point arithmetic units, as available on every desktop-class CPU.

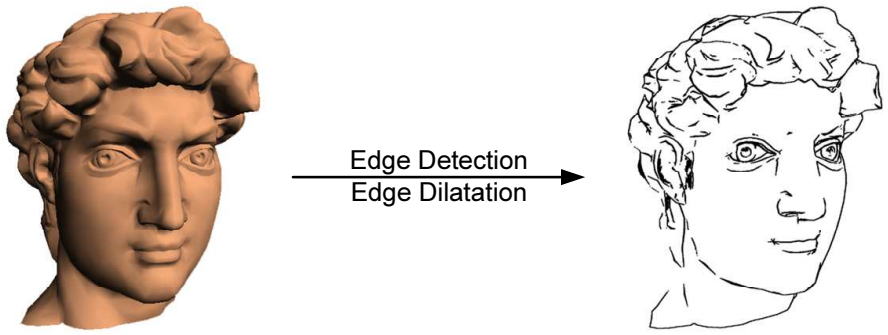


Figure 3.3: Image post processing using the G^2-Buffer framework [62].

Other limitations of mobile devices are display size and resolution. Upcoming mobile phones, e.g., will support increased resolutions of up to 480x800 pixels[1], but the diagonal of physical displays will remain smaller than approximately four to five inches; thus, there is a natural limit of resolution where humans can no more resolve the pixels. Battery capacity is a further limitation where the development process is slowly but very important as more computation power, brighter displays, and improved wireless connectivity are possible with the availability of more electrical power.

In general, applications running on a mobile device—especially interactive graphics—have to respect these limiting aspects in order to be applicable. Future mobile devices will have increased GPU/CPU performance, an improved display resolution, and probably a longer power-up time, but the devices' physical dimensions will stay the same. Thus, applications and visualizations still have to be adapted according to properties of the context in which they are used, e.g. the limitations of mobile devices.

3.2 High-Level Graphic APIs

For programming graphics processing units the aforementioned low-level APIs are available: OpenGL and DirectX. Both offer a fine-grained control of the rendering pipeline, which on the opposite implies that an application programmer is responsible for correct and efficient usage of the pipeline. Several abstraction layers were introduced to both improve and simplify the programmability of low-level APIs, and/or to provide APIs for different fields of application, e.g. 2D graphics. Some of these high-level APIs further integrate important graphic concepts, e.g., to efficiently manage huge 3D scenes, like hierarchical scene definition, spatial subdivision, culling, etc.

3.2.1 Scalable Vector Graphics

SVG [70] is an open standard that defines 2D vector graphics—drawings, illustrations, graphs, etc.—using a XML schema. SVG is designed for illustrations within websites to overcome the

[1] HTC Touch Diamond2: http://www.htc.com

limitation of bitmap images and allow high-quality drawings. Therefore, most modern internet browsers natively support rendering of SVG content, also Microsoft started to support SVG in their upcoming browser version Internet Explorer 9.

More important, SVG starts to become a de-facto standard to exchange 2D vector graphics. Especially on mobile devices, where 3D graphics and corresponding APIs are often unavailable, SVG offers the possibility to provide advanced graphics. In addition to internet browsers also dedicated viewer applications exist that are able to interpret and render SVG content, even on smart mobile devices. For sophisticated, animated SVG graphics a high-performance rendering system is required, preferably hardware accelerated. In fact, some mobile devices support hardware accelerated SVG rendering—based on the low-level API OpenVG [113]—although no full 3D capable GPU is embedded.

Therefore, SVG is an alternative approach to implement interactive (2D) graphics on mobile devices. In addition, SVG is available on various devices and operating systems enabling cross-platform development, research, and evaluation on, e.g., PDAs and larger-screen projection systems simultaneously.

3.2.2 Scene Graphs

A scene graph is a generic concept to organize data of a 3D scene—i.e. objects, materials, transformations, etc.—using a hierarchical tree structure for efficient processing and rendering. One of the first hardware accelerated (via OpenGL) scene graph implementation was IRIS Inventor [190]; today Nvidia's NVSG [150] and OpenSceneGraph (OSG) [155] are probably the most advanced and active projects that develop a state-of-the-art scene graph API. Although the basic concepts of both are similar, the following resembles details specific to OSG as applications developed in this thesis makes use of this API.

A scene graph defines multiple nodes types, like transformation, geometry, switches, etc.; instances of these are connected in a tree structure to form a scene graph. Figure 3.4 illustrates an example scene graph consisting of three geometry nodes (a plane, a box, and a sphere) and three transformation nodes.

The scene graph is traversed to generate the final result; thereby nodes along the path to a leaf iteratively change parameters of the rendering pipeline. Following the left-most path from the root to the sphere geometry in Figure 3.4, the first *Transform* might represent a translation and rotation; the second *Transform* translates the sphere above the box and is relative to all previous changes along the path. Optimization of the rendering performance is a primary purpose of scene graphs. Different implementations apply different optimization approaches, e.g. culling of subtrees that are not visible in the final image. As OSG is tightly coupled with OpenGL, also OpenGL-specific performance optimizations are addressed: State changes, i.e. changes in the setup of the rendering pipeline, are often major performance penalties of complex OpenGL programs and therefore have to be minimized. Thus, geometry that can be rendered using the same pipeline setup should be drawn together. OSG supports this by providing so-called *StateSets* which define partial pipeline setups. When traversing the graph, OSG can combine geometry that makes use of the same StateSet configuration and render it together to improve the overall performance.

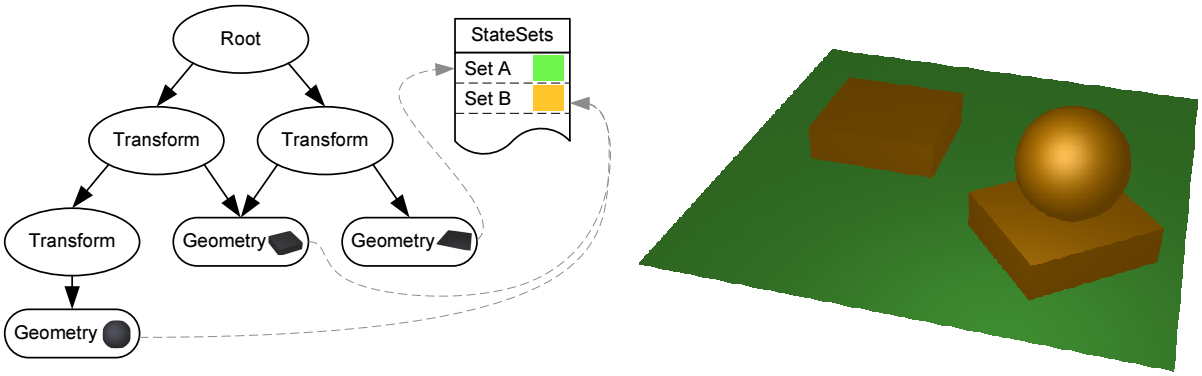


Figure 3.4: A scene graph consisting of three transformation nodes and three geometry nodes. On the left the graph structure is shown and on the right the resulting rendering. Note that the second geometry is referenced by two different parent nodes.

Changes or adjustments of a 3D scene represented as a scene graph also requires traversing the graph until the desired node or node type is found. OSG supports a concept of *nodevisitors* to automatically handle and control the traversing of the graph. A common nodevisitor is the updatevisitor that is automatically executed before each frame redraw to report the elapsed time and trigger actions like scene animations. Additional information and details about OpenSceneGraph is available at [155] and in the *OpenSceneGraph Quick Start Guide* [135].

3.3 The Visualization Pipeline

The visualization pipeline was introduced by R.B. Haber and D.A. McNabb in [86] in order to use rendering functionality to effectively present (scientific) data. Visualization is the task to transform raw, usually numeric data values to a visual representation that makes the data easier perceivable by humans. This fundamental pipeline is shown in Figure 3.5 in a slightly modified version: The commonly considered *user interaction* which controls the behavior of the three stages has been replaced with a more generic *context-aware configuration* behavior. Thus, this thesis considers context-aware visualization techniques that adapt not only according user interaction but also based on generic context aspects like device type, display size, data amount, location, orientation, etc. Thereby, user interaction is seen as a part of the generic context.

The visualization pipeline as defined in [86] and shown in Figure 3.5 is composed of three stages: filtering, mapping, and rendering. All data that is to be visualized has to be piped through these stages that transform the input data to the final image.

Filtering prepares the incoming raw data to generate visualization data. Within this step various operations are possible, e.g.: selection of a data part, smoothing of the data values, interpolation/resampling of the input data, denoising, or segmentation.

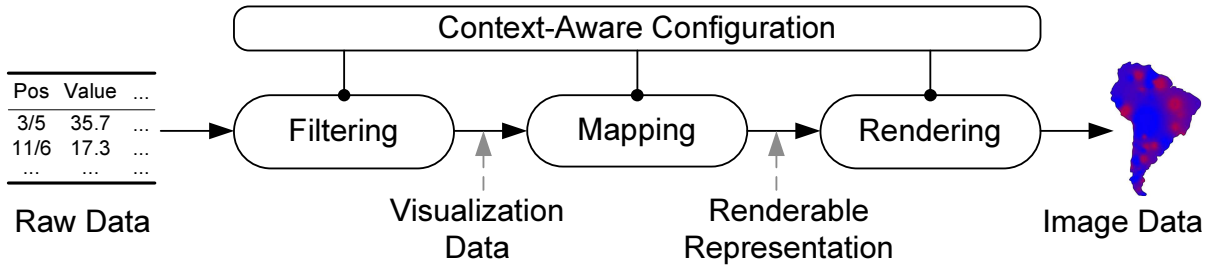


Figure 3.5: The Visualization Pipeline.

Mapping is the fundamental part of each visualization technique: it maps the visualization data to a renderable representation (visual elements). These may be geometric primitives, color, and texture that thereafter convey the content of the visualization data.

Rendering generates the final result of a data visualization by projecting the renderable representation onto an image. Thereby, the rendering stage makes use of the previously introduced rendering pipeline. Often, this final step of the pipeline outputs its result directly to a display to show the result to the users.

All stages of the pipeline are adjustable and have to be tuned to achieve appropriate results. Thereby interactive visualization, i.e. techniques that provide instant feedback on changes of parameters, is most important as often users manually fine-tune parameters while observing the resulting image. Furthermore, users tend to interact with the visualization in a way that different viewing positions and angles are chosen in order to understand (3D) spatial structures of the data. Such interaction often influences only the final rendering step of the visualization pipeline which is therefore important to perform at interactive rates in order to provide instant feedback to the users. However, as this thesis widens the concept of user interaction with the visualization pipeline to generic context-aware configuration, other fast changing aspects—e.g. a reduced network bandwidth or changing lighting conditions—intensify the demand of interactive visualization solutions.

3.4 Virtual and Augmented Reality

In addition to using computer graphics for information visualization, it can even be used to represent or simulate an entire artificial environment, termed as *virtual reality*, or present information in the context of a physical environment, referred to as *augmented reality*. Users of virtual reality systems experience a state consciousness which is termed *immersion*; although the environment is artificial it behaves as a real, physical environment. Often sophisticated hardware and installations are used to improve the immersion of VR systems.

In contrast, augmented reality embeds artificial information in a real, physical environment. Again, advanced hard- and software technology is used in order to achieve the impression of physically embedded objects. Thereby, the main challenge is to synchronize the view to the

physical environment and the virtual view, applied for the augmentations. Sophisticated tracking systems are used to provide the required high-precision location and orientation information of a viewer to which augmentations are applied. When considering location and orientation as context information—as proposed in this thesis—augmented reality visualizations are in fact rudimentary context-aware visualizations.

3.4.1 The Continuum of Reality to Virtuality

In-between both extremes *real environment* and *virtual environment* arbitrary degrees of virtualization exist, not only augmented reality. P. Milgram et al. propose the reality-virtuality continuum to define possible configurations of mixed reality setups [141]. Figure 3.6 depicts the continuum, where the right side represents the physical, real environment and the left side marks an entire artificial, virtual environment. In-between, arbitrary constellations of mixed reality are possible.

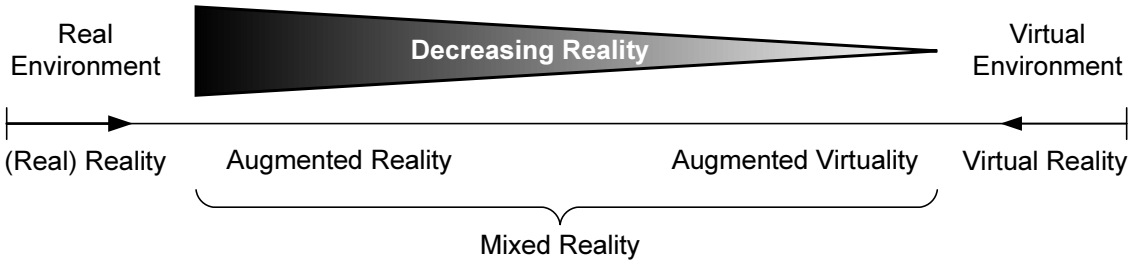


Figure 3.6: P. Milgram et al.'s reality-virtuality continuum to describe different mixed reality scenarios [141].

Based on this insight that reality, augmented reality, and virtual reality are related to each other, a prototype to conduct a small user study to investigate different modes of mixed reality was built. Results are presented in cooperation with O. Siemoneit in [59] and further discussed in Section 5.1.2.

3.4.2 Dedicated In-/Output Technology

Systems that implement VR or AR aspects require specific hardware to sense the physical environment—tracking systems, camera, user-input devices, etc.—and most often dedicated display technology—head mounted displays, projectors, etc. Tracking the user's location and viewing direction is required to adjust the virtual environment accordingly (for VR setups) and to show augmentations at precise locations relative to a physical environment (for AR setups); solutions for tracking are mentioned in Section 2.2.

Dependent on the utilized display device, augmented reality requires additional information about the physical environment, most often a live video stream is used (video-see-through). In contrast, optical-see-through display technology provides the possibility to see though the display and at the same time present information on it, thereby augmenting the real-world view with

information. In this thesis the following hardware was used for AR/VR projects: optical-see-through retina display, optical-see-through LCD HMD, video-see-through LCD HMD, camera-equipped TablePC (video-see-through), and camera-equipped PDA (video-see-through).

4

A Framework for Context-Aware Visualization

Visualization of data has the primary goal to make data more understandable to humans. Research in the area of visualization techniques continuously invents new approaches to handle the growing amount of available data and its various formats. In general, most often the design of visualization methods is based on the previously introduced visualization pipeline (see Section 3.3), but the resulting images of different techniques applied to the same data might differ vastly. Consequently, different techniques have to be applied dependent on the current situation, i.e. based on aspects the users is currently interested. This *intelligent* selection process becomes an increasing challenge with the growing variety of data formats and corresponding visualization approaches.

The goal of the thesis is to address this problem with the use of context information to provide context-aware visualizations. In this chapter a framework is proposed to provide an efficient environment for implementing, testing, and using context-aware visualizations to improve the overall benefit and usage of visualization by selecting, adapting, and combining multiple methods to optimally present the underlying data. In addition to well-established interactive visualizations, context-aware methods utilize available context information—e.g. position, orientation, or hardware capabilities—to automatically control the selection and configuration of visualization techniques to find an optimal visualization setup adequate for the underlying data. Some previous work on context-aware applications also considered visualization aspects. However, support for sophisticated interactive visualization, optionally for mobile devices, is seldom mentioned in research, but as context data is intrinsically dynamic and mobile users frequently change their context, the proposed framework primarily focuses on both aspects.

Previous work from H. Senay and E. Ignatius resulted in the Vista system [184], where the goal is to automatically generate combined visualizations from basic techniques. Several rules describe how visualizations can be combined to achieve the most pleasant results. After definition of the input data, the system tries to find a possible visualization, considering the input data types and their correlation. O. Gilson et al. make use of domain knowledge in form of ontologies to generate information visualizations from domain-specific web pages [79]. In contrast to our context-aware approach, both systems consider only the input data itself and static domain knowledge. In addition, the resulting visualizations are targeted for a single platform and do not offer dynamic context-aware presentations. A system based on scene-graph concepts

is proposed by G. Reitmayer and D. Schmalstieg [166]. They present an extended scene graph that is automatically adapted during an update phase based on specific context settings. E. Mandez et al. [138] make use of this system to develop context-driven visualization. In addition to this, the proposed framework for context-aware visualization makes use of context information to automatically find the most suitable visualization by performing a real-time reevaluation on the available techniques, which is not foreseen in the technique presented in [166]. A conceptual framework for context-sensitive visualization with focus on recognition and handling of higher-order context is proposed by E. Jung and K. Sato [106]. Their approach is envisioned for interactive applications, but does not support interactive visualizations or real-time context-aware adaptation.

Further, a number of specific prototype applications and frameworks exist. F. Shibata et al. propose a framework to develop mixed and augmented reality (AR) applications [186]. Thereby, they focus on location awareness, as typical for mixed-reality applications, and computation load balancing for AR applications. Context-aware selection or configuration of visualization approaches is not considered. The development of a context-aware tourist guide is presented by K. Cheverst et al. [36]; they propose to change the presentation according to context aspects like location history or user interests. However, their proposed system is targeted for static non-interactive presentation of information whereas this thesis focuses on interactive visualizations. An interactive prototype for a mobile location-based information presenter is proposed by S. Burigat and L. Chittaro [29]. Their tourist guide application (LAMP3D) renders 3D VRML models interactively while considering the location context; further context aspects are not analyzed. J. Rekimoto and K. Nagao already present some context-awareness aspects in their prototypes, but also do not focus on sophisticated visualization or context-aware presentation [168].

In contrast, the proposed generic framework for context-aware visualization—developed in collaboration with D. Weiskopf [63]—focuses on sophisticated interactive presentation, especially on mobile devices. Several requirements have to be fulfilled by such frameworks that strongly affect the resulting design. In addition to the usage of context information for visualization selection and configuration, also a basic approach for a context-aware user interface is discussed.

4.1 Requirements for Context-Aware Visualization

When dealing with arbitrary context aspects, assumptions on data type, data amount, or the rate of change cannot be made. The definition of context aspects—i.e. position, direction, time, status, etc.—varies dependent on the field of application. Therefore, a framework must not place any restrictions on these attributes and has to support arbitrary, even unknown context properties. An easy integration of further visualization techniques to handle emerging context definitions, previously not considered by the framework, is important to the same degree. In addition, research on visualization continuously improves existing techniques and finds new ways to present data, probably utilizing specific hardware features, for which the integration has to be feasible.

Besides the ability to reuse visualization modules for multiple applications—similar to web pages that make use of the Google Maps module to visualize map-based data [81]—a major

purpose for realizing a framework is the automatic selection of adequate visualization techniques, based on context information. As multiple alternative approaches might be available to render the data, the system has to evaluate and rank the various methods for their appropriateness within the related context to find the most adequate approach. For multiple data attributes, optionally originating from different sources or given in different structures, a combination of multiple visualization methods to support joint visualizations is required. This way, data can be shown in context within other related data to support users in understanding the data, which is often termed as focus&context approach as discussed in Section 5.1. To account for fast changes in context aspects, e.g. position or viewing direction, visualization techniques have to be capable to render at interactive frame rates. Currently, most interactive methods utilize graphics hardware to satisfy the demand of huge computational power [189, 203]. Using hardware support also helps to process huge amounts of context-annotated data efficiently, which will probably be available when data sources like Nexus are queried.

For real-world scenarios, where often mobile client devices are used to view the resulting illustrations, a sophisticated preprocessing has to take place in order to support interactive visualizations on these computation-limited devices. Most existing solutions perform such preprocessing on a server component of a network-connected client-server architecture. Thereby, the network connection is essentially a wireless connection with limited bandwidth, e.g. IEEE 802.11-WLAN or IEEE 802.15.1-Bluetooth. Special considerations have to be made to minimize the required network bandwidth, e.g. using an adequate data compression scheme, and to cope with communication latencies to efficiently support mobile clients. Further, different levels of preprocessing may be supported so that either final images are transmitted to the clients or data structures including rules to compute the visualizations on the client devices. If multiple concurrent mobile clients are active to, e.g., examine the result of a visualization in collaboration, server-integrated caching strategies and per client-type data preprocessing will further minimize the throughput demand of the infrastructure network.

Naturally, user interaction for such context-aware interactive visualizations has to be considered, especially for mobile-device scenarios. On desktop workstations common user-input techniques are based on keyboard and mouse. For mobile, context-aware scenarios more advanced approaches are needed. Often hands-free operation of mobile applications is desirable as, e.g., proposed by S. White et al. for augmented reality [209]. Hands-free interfaces are often realized using sensor data—in [209] head-movement is used to trigger user events—which the framework already deals with in terms of context information. Further, an adaptation of the user interface according to the context is possible to, e.g., resize the graphical user interface elements. Although this thesis focuses on techniques for context-aware visualization, some aspects of context-aware user interfaces are examined, as they are beneficial for context-aware mobile applications.

In summary, a framework for context-aware visualization techniques has to primarily address the following demands:

- Handling of unknown context/data structures and attributes.
- Easy integration of novel visualization approaches.
- Automatic selection of adequate techniques for processing and visualizing the underlying data.
- Combination of multiple visualization techniques to generate joint visualizations that concurrently present multiple data sources.
- Support for graphics-hardware assisted visualization techniques to support near-realtime changes of context aspects.
- Efficient processing of large amounts of context-annotated data.
- Support for multiple, network-connected clients using adequate data representations.
- Fast responses to changes in context even for network-connected clients.
- Support for sensor-/context-based user interaction and context-aware user interfaces.

4.2 Framework Design

A framework that fulfills the above mentioned requirements has been developed as part of this thesis. The system is tightly coupled to the Nexus system, which is used as an underlying context-data management system. The design is divided into several functional modules to support different configurations that are useful for distributed installations or mobile-client scenarios. Thus, it is also possible to combine the presented framework with other context-management systems [11, 175, 47]—as long as hierarchical data definitions and queries to the context data are supported—only by replacing the communication functions for the context-data handling.

Besides the communication with a context-aware basis framework, central modules are: preprocessing of context data, searching and selecting an adequate visualization technique via data-definition matching, and preparation of visualization data to support efficient hardware-accelerated visualization techniques. A schematic overview of the realized system is shown in Figure 4.1.

The left part of Figure 4.1 shows the utilized context-management framework Nexus, whereas the two most important aspects are the context-aware data and its XML-based hierarchical data-type definition. Both are consumed by the central component, shown in the middle section. A further database, shown in the bottom part of Figure 4.1, for storing the integrated visualization techniques is continuously queried during the visualization matching process. The resulting visualization is prepared and configured, dependent on the context information, and transferred to the client application (see right part in Figure 4.1).

The design of the framework was made with respect to interactive visualization methods;

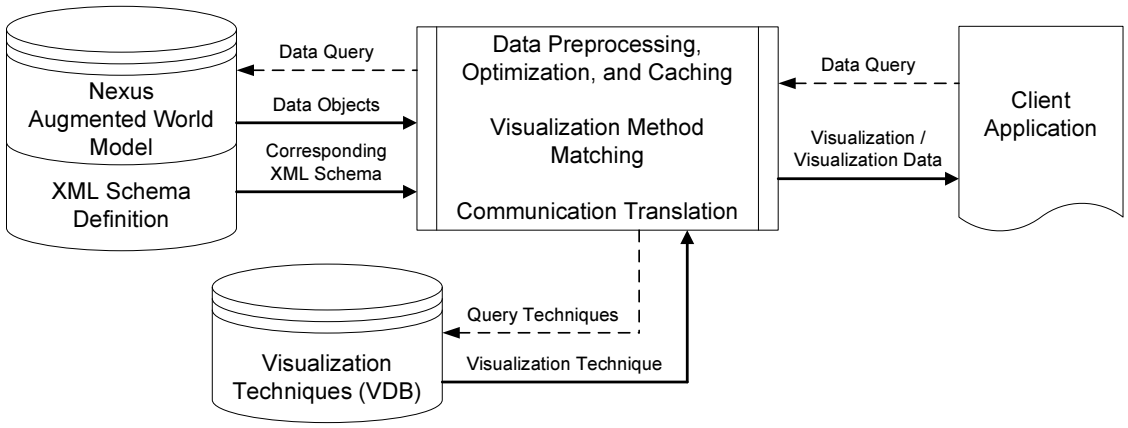


Figure 4.1: Schematic framework overview responsible for communication, preprocessing, and visualization technique matching.

therefore, data structures and algorithms are realized using graphics-hardware friendly concepts.

4.2.1 Graphics Hardware Support

A high-level API is used to provide graphics hardware support within the framework. The open-source scene-graph API *OpenSceneGraph* is based on the OpenGL graphics programming API [183] and is platform independent, which allows installing the framework on various operating systems. Consistent with the concepts of scene graphs (SG)—as discussed in Section 3.2.2—an instance of a visualization technique is a sub scene graph, which can be attached to a higher-level graph to, e.g., combine multiple visualizations.

The techniques themselves are thereby not limited in their implementation and can freely choose to use CPU, GPU, or other any other available computing resources to generate the visualization. However, the implementations have to be consistent with the structure of SG nodes, so that mechanisms of the SG like *update* or *draw* still work appropriately. However, if a client device cannot meet the hardware requirements, e.g. does not provide hardware-accelerated rendering, a fallback utilizing server-installed hardware to generate the visualizations is possible.

The benefit of this architecture is that implementations of context-aware visualization techniques are motivated to utilize GPUs for achieving interactive frame rates. Furthermore, existing OpenGL-based or even OpenSceneGraph-based implementations of visualization approaches can easily be integrated to increase the variety of available techniques [107, 21]. Even more interesting are projects that integrate existing visualization packages into OSG as shown for VTK [182, 115] in [22] by K.J. Blom.

With this approach, interactive renderings of context-aware data are possible, but for most techniques the interactivity is limited to specific context attributes: position, orientation, and probably some technique-specific configurations. More drastic changes in the context, e.g. selection of different object types that have to be visualized, are hard to support at interactive feedback rates. Such events involve a query of further context-annotated data, preprocessing, selection of an adequate visualization, and the generation of visualization scene graph nodes.

Already a query for additional data executed by the utilized Nexus framework prevents interactive feedback rates; therefore, caching and preprocessing strategies are necessary within the developed context-aware visualization framework.

Minimizing feedback delay to the client application, if additional context data has to be queried, implies a caching of generated visualization nodes. This overcomes the most time consuming process of data query and selection of appropriate visualization methods. The framework simply keeps an in-memory copy of generated GPU-friendly visualization nodes. If client applications request a visualization of a data object that has already an in-memory visualization instance, the corresponding sub scene graph is simply transmitted in the same way as newly generated visualization instances. A compact binary representation is used to transfer visualization instances including their required data, realized by an extension of SG nodes for binary serialization.

4.2.2 Processing and Communication Infrastructure

A system that supports heterogeneous mobile clients has to cope with limitations of these devices. Computation power, network bandwidth, and limited storage are the most severe ones. The proposed client-server system is designed to address these aspects. It supports processing of arbitrary (context-annotated) input data and is based on a caching and preprocessing concept to prepare the data based on the client's requirements and further context attributes. The server acts as a communication endpoint for heterogeneous client devices and optimizes the wireless transmitted data (Section 4.4); it is further designed to work with multiple concurrently connected clients in a way that data preprocessing is only done once and cached for all clients. For the processing of data, queried from a federated data source, an appropriate technique is selected, based on the user's context, so that the resulting visualization data can efficiently be handled by client devices. The necessary per-client context is made available via an aggregation of data provided by the context-aware Nexus framework and data from local sensors of the connected client. In order to support up-to-date real-time context information, both sources are synchronized to build the per-client context that is consistently available on the server and client. Aspects that are included in the context are amongst others: position, view direction, device type, device screen resolution, occupied screen space, network connectivity, etc.

For the communication with the underlying context-aware platform a descriptive communication protocol is employed. In Nexus, XML/SOAP requests are used to access the federation component, but such protocols would be inappropriate for direct communication with mobile client devices. This kind of communication tends to increase latency—due to complex structure parsing—and the total amount of data transmitted, especially if large datasets have to be supported. In contrast, the server-based preprocessing—as illustrated in Figure 4.1—allows preprocessing the raw data once and storing it in a cache using a data structure that is more appropriate for visualization purposes and allows efficient communication to client devices. Therefore, subsequent queries to the same data objects—possibly from a different client—benefit from cached intermediate precomputation and experience a faster server response. In doing so, the underlying context-aware framework does not have to retrieve and federate data from possibly slow, remote providers.

The proposed generic system supports multiple techniques to process data objects of the same or different type and amount via a plugin-like system. In the following, data processing for visualization on heterogeneous mobile and desktop computers is discussed, but the proposed client-server system may also be used to perform other not necessarily visualization data processing.

4.2.3 Visualization Technique Templates

Key components of the framework are the visualization techniques themselves. The server has—in addition to the data queried from the AWM—access to a database of visualization methods (*VDB*) that implement different visualization techniques, as depicted in the bottom of Figure 4.1. Thereby, visualization methods are implemented via separate plugin-like templates that can even be added during runtime. Templates can access any resource and even spawn additional threads to, e.g., allow parallel preprocessing tasks. Each individual visualization template, available through the *VDB*, is required to implement the following four functional parts:

1. A function to query the template for appropriateness to visualize a specific data type by providing the corresponding data description schema.
2. Processor to handle raw context-data objects, provided by the utilized federation.
3. Implementation of the visualization technique, realized by a sub scene graph.
4. I/O routines to transmit/receive all required data of the template's visualization.

The appropriateness weighting function (1) is used by the visualization matching, detailed in Section 4.2.4, to weight each available technique according its ability to present a specific data type considering the current context condition. The framework provides information about the data types based on which the function returns a normalized scalar value, rating the appropriateness to visualize the data.

After an evaluation of all available techniques (see Section 4.2.4), the matching instantiates the most appropriate techniques—i.e. at least one but often multiple alternative approaches—and provides the instance(s) with data through the defined interface (2); thereby, enabling each activated visualization to prepare and store the visualization data in a plugin-specific way, optionally including the steps filtering, mapping, and even rendering as defined in Section 3.3 by the visualization pipeline. Already at this early processing stage unneeded data can simply be skipped and does not consume further storage or processing resources. Any additional data required by the template to generate an instance of a visualization technique is also retrieved. This may include icon images, proxy geometry (used to show information), but also additional context-aware data stored in Nexus. Furthermore, external data referenced by Nexus data objects is accessed and retrieved, if needed by the applied technique. Huge complex, possibly proprietary, data types, not represented in Nexus can be integrated this way, as discussed in Section 2.1.4.

Instances are initiated by plugins themselves and can therefore be generated considering the current context, e.g. available client hardware or user preferences. Sophisticated visualization

techniques might require dedicated hardware support, but if the context information shows unavailability, a simpler fallback approach is instantiated. The number of instances generated is also specific to each template: Simple techniques generate a visualization instance for each received data object, resulting in individual visualizations for each value. More advanced methods try to group data objects to generate a combined visualization for a number of values. This behavior is an important aspect as individual visualizations cannot adequately represent, e.g., continuous scalar data files as shown in Section 5.3.1. Preprocessing and implementation of visualization techniques are tightly coupled with the underlying scene graph; therefore, visualizations are represented as partial scene graphs with possibly template-specific node extensions (3). After preprocessing, a template returns the corresponding sub scene graph; that can be executed to calculate the final visualization dependent on the client and its current context, which is further detailed in Section 4.2.4.

Dependent on the client capabilities, the resulting intermediate binary data can therefore be: raw data objects, standard scene-graph nodes, extended scene-graph nodes, or even images streams transmitted via a (wireless) network connection to the client (4). Functions for serialization (server) and deserialization (client) of an instanced visualization are provided by each visualization template. In general, a visualization is implemented via a sub scene graph with several data objects attached, e.g. 2D/3D textures, GPU shader code, etc. This entire information has to be transferred to the client application to execute the visualization within the application. As each template individually extends scene-graph nodes and stores data specific for its visualization, the serialization functions have to be provided by the templates themselves. This implies that client applications have to utilize the same template in order to be able to deserialize the received visualization. The introduced dependency on client-side availability of plugins can be eliminated by the framework with the following compromise: If a specific visualization plugin is only available on the server side and the techniques makes use of non-standard scene-graph nodes, then the clients cannot process the template-specific visualization data and the framework reverts to basic image-stream transmission, using techniques discussed in Section 4.4. An alternative option is that the server transmits the binary plugin that matches the client's architecture specified within the context data. This way, client applications can automatically be updated to benefit from new visualization techniques to display the heterogeneous data encoded within the transmitted scene-graph structure.

4.2.4 Technique Matching utilizing XML-based Type Definitions

The automatism to select and configure appropriate visualization techniques strongly makes use of hierarchical data definitions as, e.g., provided by the Nexus XML schema (the so-called AWS [145]), where all data objects have to be defined. The design of the schema makes use of *substitutionGroup="..."* attributes, *<restriction base="..."/>*, and *<extension base="..."/>* tags to compose extended object classes and attributes in a hierarchical manner. The following listing shows the description for an URL attribute type, that can be used to define an attribute of a class object within the schema.

```
<simpleType name="NexusUriType">
  <restriction base="string"/>
</simpleType>

<complexType name="NexusUriAttributeType">
  <sequence>
    <element name="value" type="nsat:NexusUriType"/>
    <element ref="nsas:meta" minOccurs="0"/>
  </sequence>
</complexType>

<element name="uri" type="nsas:NexusUriAttributeType"
  substitutionGroup="nsas:NexusAttribute"/>
```

Listing 4.1: Hierarchical definition of an URL attribute, excerpt of the Nexus AWS [145].

Note that the resulting *uri* attribute can be inserted wherever the schema requests a *NexusAttribute* as it is part of this substitution group. Furthermore, it is based on (i.e. derived from) the basic *string* type. This hierarchy is fundamental to the automatic selection, as for data types for which no corresponding processing template is available the system can iteratively cast the data to its base type and try to find matching techniques until a basic type, e.g. *string*, is reached. This ensures that all, even future data elements of the AWS can be visualized with the limitation that newly added, specific attributes might not be shown in the most optimal way, if only a visualization for the base data type is available.

The entire evaluation process to find possible techniques in the *VDB* that match the requested data type is implemented via a sequential process, as depicted in Figure 4.2. The illustration shows the data flow of each individual data element and the resulting visualization, transmitted to the clients. For each element of the data-object stream that is piped into the matching process originating from the augmented world model, all available visualization techniques are queried to rank their usability to visualize the requested data type. Each template of the *VDB* returns a normalized scalar value to reflect the quality/appropriateness q_i of the resulting visualization for the given type. For instance, if scalar values with 2D location information are received, a technique that handles full 3D location information is less appropriate than an approach designed for data given in 2D. Techniques with $q_i > 0$ are assembled to a list of possible visualization-technique candidates. For data types where no corresponding processing technique is available—i.e. all available templates return $q_i = 0$—the automatic selection mechanism makes use of the fundamental hierarchical structure of the data type descriptions: The system iteratively casts the data objects to their base types and uses those in replacement to find matching techniques until at least one technique is found and inserted into the list of possible techniques. As the attributes and classes defined in the AWS are based on only few basic types (*string*, *integer*, *float*, etc.), and for each of these a basic visualization plugin is available, it is guaranteed that the system can process *all* data, provided by the AWM. However, the quality and usability of the result is still strongly dependent on the utilized visualizations.

As the system is designed to work with data streams, to allow processing of huge datasets that are streamed from a database without the requirement of reading the entire data at once,

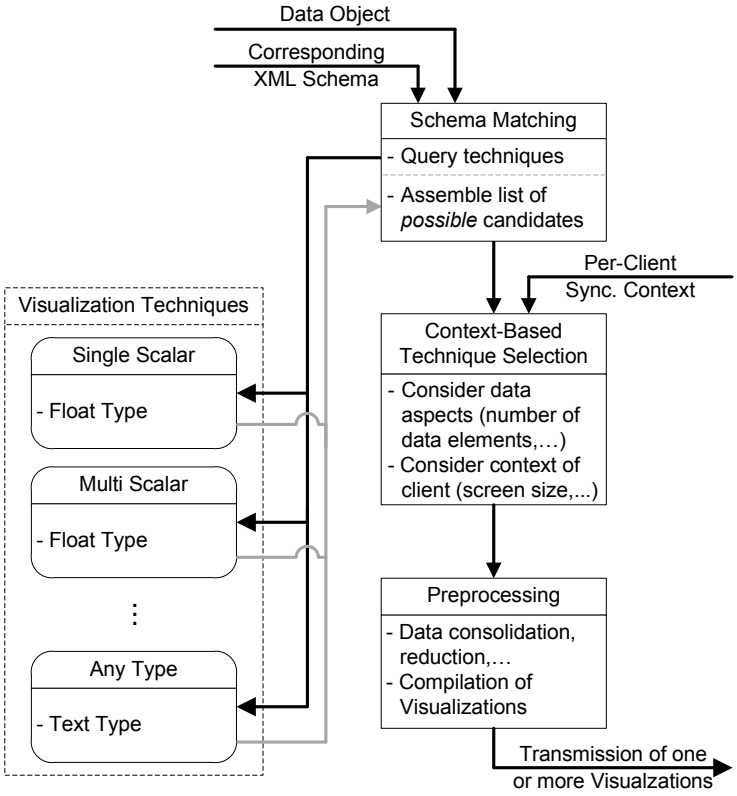


Figure 4.2: Data-flow path during the evaluation process for relevant visualization techniques using XML-based data type descriptions.

the amount of data of a single entity type and also further possibly related data objects are not a priori known. However, content-related data, i.e. the amount of data, spatial distribution of the individual data elements, and possibly other aspects might be relevant to the selection process of appropriate visualization techniques.

To consider these aspects in the selection process, all techniques in the list of possible visualizations are provided with raw data from the stream of data objects, generated by a query to the AWM. Thereby, the individual techniques only store data as long as they would return an appropriateness q_i greater than 0.0, otherwise any data provided by the framework is discarded. This way, techniques that are only appropriate for visualization of few data objects can simply check with each received value if already too many objects are read and then skip any subsequent data values.

After all data from the query is steamed to the potential visualization templates, a combination of per-client context information and properties of each technique—e.g. the amount of data objects the method can visualize simultaneously or the spatial density of the data distribution suitable for this approach—are used to decide which visualization is finally selected as most appropriate for the current situation. This process—summarized as *Context-Based Technique Selection* in Figure 4.2—again makes use of an appropriateness weighting function, but this time also aspects of the data content the visualization has received are considered.

4.3 Context-Based Configuration and Rendering

The template matching and selection process described in the previous paragraph considers mainly static context information, i.e. data type or amount of data items; thus attributes which do not change frequently. However, real-time or at least interactive visualization that depends on a possibly dynamic, fast-changing context—e.g. mobile users changing their position—has to support a fast update mechanism to adapt visualization parameters or even change the entire technique. Thus, visualizations might change although no additional data has been queried from the AWM.

The necessary per-client context is made available via an aggregation of data provided by the framework and data from local sensors of the connected client. In order to support real-time context information, both sources are frequently synchronized to build the per-client context that is consistently available on server and client.

In general, the entire selection process can be repeated, if a context change is detected; this, however, will introduce a high latency for adaptations due to the costly schema matching. An efficient reconfiguration according to varying or even highly dynamic context aspects requires a continuous adaptation and reevaluation of the utilized techniques.

4.3.1 Adaptation of Instanced Visualizations

High-frequency but moderate changes in the context, e.g. position or view direction updates, are often handled via parameter adjustments of the utilized visualizations. This requires, in addition to the previously presented framework design, a dynamic adaptation and selection component to support near real-time changes of visualization parameters or even technique switches.

Virtually all scene graphs support the concept of update calls to their nodes in order to allow the scene-graph nodes to adjust/animate themselves. Nodes in OpenSceneGraph support the general concept of UpdateVisitors as explained in Section 3.2.2. As context-aware visualization techniques are implemented as sub scene graphs, each technique automatically supports this scene-graph traversal interface. The universality of OSG's visitor concept allows utilizing multiple visitors for different tasks. Thus, for the purpose of context update, the framework implements a specific node visitor called $ContextUpdateVisitor$ that is triggered whenever the synchronized per-client context is changed. It traverses all instantiated visualization nodes and reports the updated context via the aforementioned node visitor interface. For adaptation, each instantiated visualization implements a custom update function—similar to the appropriateness weighting function of templates (see Section 4.2.3)—to change parameters according to the per-client context provided by the $ContextUpdateVisitor$. Again, for adaptation of the visualization's parameters unrestricted access to all context aspects is available; evaluation of various context aspects and resulting adjustments is detailed in Section 4.3.3. The weighting function further reflects the appropriateness to visualize the attached data regarding the updated context by returning a normalized scalar value, which is important for switching of techniques as discussed in Section 4.3.2. This way, visualization techniques can adapt themselves to the changed situation and respond to it on following requests.

In principle, the visitor is executed by the server to update instanced and cached visualizations. For mobile low-performance client devices, i.e. devices that receive an image stream of the resulting visualization from the framework, this adaptation directly affects all images received after the context update. Further, the synchronized per-client context automatically transmits context changes detected by client's locally attached sensors to the server. In contrast, more advanced clients that are able to execute the visualization scene graphs locally are unaware of server-initiated context-aware parameter adaptations. Therefore, such clients have to execute a local $ContextUpdateVisitor$ analogous to the server.

4.3.2 Fast Technique Switching

After a client queried the framework for visualization of a dataset, the framework acquires the data and builds a list of possible visualization techniques for the received data types (see Section 4.2.4). The selection thereby considers the context as available by that time, future changes in the context are handled by the techniques themselves as discussed in the previous section. However, if the context has changed tremendously, the previously selected visualization technique may no longer be adequate to visualize the considered data and a fast change to a more suitable visualization is required.

Therefore, the entire adaptation mechanism has to include a context-aware switching of multiple alternative techniques that handle the same data type for different context situations properly. The framework integrates special support for fast switching of visualization techniques: All visualizations that are assembled to the list of possible techniques during the initial template selection phase are instantiated and attached to a context-sensitive switch node, which is basically a context-aware group node.

In consistency with the update mechanism for visualizations, context-sensitive switches also receive dynamic context updates using the same $ContextUpdateVisitor$ as used for configuring the techniques themselves and can perform a context-aware selection of attached techniques without a noticeable delay. An exemplary graph structure of a context-aware switch node for runtime selection of alternative techniques is illustrated in Figure 4.3.

In contrast to a standard scene-graph switch node, where child nodes are explicitly turned on or off, the proposed system allows various types of context-aware switches. The simplest version just reevaluates the appropriateness q_i of its attached visualization nodes, triggered by the $ContextUpdateVisitor$, and activates the most appropriate one for the currently active per-client context. Figure 4.3 illustrates a subgraph for data that can be visualized with a diverse set of methods, which are selected based on context information. Still, for drastic changes of the context, e.g. a changed display device, a server-side restart of the entire selection process is needed.

As described so far, the selection and configuration of various visualization techniques is based on parallel instancing of multiple techniques. This implies that for advanced clients, which are capable of performing visualizations themselves, either retransmissions of newly selected visualization methods are required or multiple techniques including a context switch have to be transmitted during the initial visualization setup. This might result in an increased data amount to be transmitted, dependent on the number of available visualization approaches for corresponding

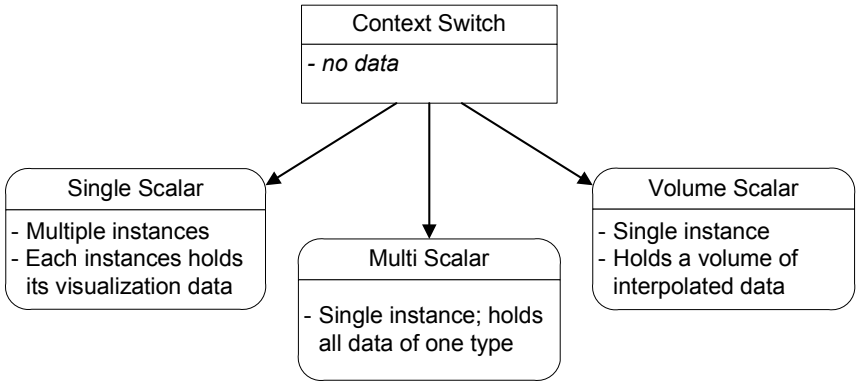


Figure 4.3: Context switch to dynamically switch between applied visualization techniques: Multiple alternative visualizations for a specific data type (here: *Single Scalar*, *Multi Scalar*, and *Volume Scalar*) are instantiated and available for execution to support low-latency visualization technique changes.

data format and their storage requirements, introducing a noticeable delay before a visualization is rendered on the client device. To overcome this disadvantage, a far more efficient technique—in terms of network bandwidth and response time—has been developed.

Lazy Instancing

The concept of the proposed context-aware visualization framework supports fast context-aware changes of visualization techniques by switching between a preselected number of possible visualizations. Therefore, not only a single instanced visualization method is transferred to clients as a result of the schema-matching process but multiple instances. This way, technique changes are fast, but at the expense of an increased delay for the first visualization results.

An optimization for this approach is to submit only one instanced visualization and further references to possible alternative visualization techniques, adequate for the corresponding client with respect to context and data type. References are basically templates that did not yet receive raw data for visualization-friendly preparation and therefore consume minimal memory. Clients can then instantiate alternative visualizations on demand, but require—compared to the previous approach—the raw data in order to instantiate visualization templates. As the raw data can be shared between all appropriate not-yet-instantiated visualizations, the context-switching node provides a container to store this data. This lazy instancing of visualizations further helps mobile clients to minimize memory consumption, which is often an extremely limited resource. The resulting data structure on the client device is illustrated in Figure 4.4.

Context switches that reference uninstantiated visualizations have to trigger the instancing procedure of templates and provide necessary raw data, if the corresponding technique is to be activated due to a change in context. Depending on the technique the subsequent processing may introduce a noticeable delay during the switching process; however, users only have to wait for a single preprocessing time versus the time for preprocessing and transmitting all techniques during initialization.

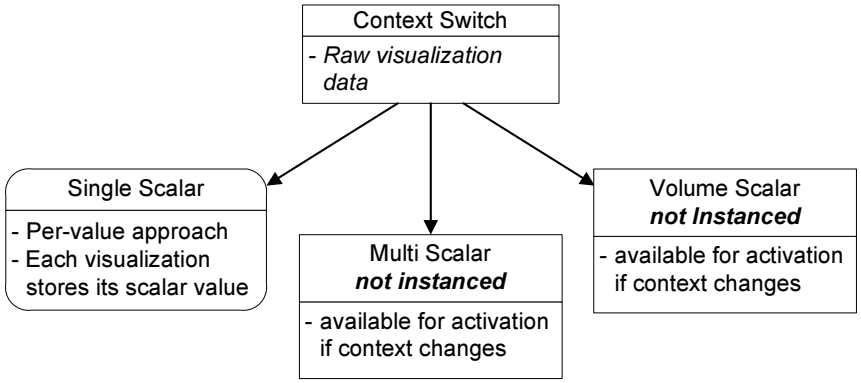


Figure 4.4: A context switch that references uninstantiated visualization prototypes. Triggered by changes in the context, templates are instantiated on-the-fly using the raw data provided by the context switch.

For repeated changes in context, the response for technique switching is improved by a caching strategy. If a change in the context information forces a context-switch object to select another visualization technique, the previously selected one does not directly get deleted and is kept in memory for later use. This caching optimizes situations where the system frequently reverts to a previous visualization at a later point in time. To keep the client-side scene graph at a reasonable size a further node visitor frees cached visualization-technique instances based on a last-used timer, thereby balancing response time versus memory consumption.

Still, this approach is not optimal as raw visualization data might be larger than preprocessed visualization data. A hybrid approach might be beneficial: raw data could be processed to generate condensed intermediate visualization data that is common for multiple techniques. This common intermediate visualization data can then be transmitted to the client and used to replace the context switch's raw visualization data, shown in the upper part of Figure 4.4, to minimize data transmission time and memory consumption.

4.3.3 Evaluation of Context Aspects

Weighting of individual context aspects for the evaluation of visualization appropriateness is flexibly implemented by each template and context switch in a way that access to all attributes of the per-client synchronized context is available, e.g. position, view direction, screen size, etc. Based on these values the technique adapts its parameters resulting in an adequate output.

The most fundamental adaptation of a visualization is the configuration of a virtual camera, used by computer graphics algorithms for projecting (3D) data onto the 2D display. Primary parameters of a camera are location and orientation; both are usually manipulated via a user interface. In context-aware systems these are just additional aspects of the entire context of a (mobile) user and can therefore directly be utilized to adjust a view matrix (see Section 3.1). Using sensors to capture and provide a user's position and view direction as context information naturally allows visualizations to match the user's view resulting in augmented reality applications. On top of that, AR is often used to highlight specific features/objects of the physical

environment; if the quality of location or orientation information is available, it can be used to configure visualization parameters like an adjustment size of displayed location markers. If applicable, distance to a point of interest (POI) may influence the level-of-detail used to visualize information related to this POI. Further, the number of POIs that are currently visible to the user may also have an influence to the visualized detail. This way, visualizations may show at most two POIs in highest detail, those that are nearest to the user, maximal three at medium detail, the remaining up to 5 POIs at minimal detail. This shows that already with the evaluation of few context attributes, visualizations are able to flexibly adjust to the current situation resulting in much more pleasant results. Further context aspects that are used to control visualization results are mentioned in Chapter 5, where concrete realizations of context-aware visualizations are discussed and prototype results are presented in Chapter 6.

In contrast to visualization instances, context switches can query its child visualizations to retrieve their scalar appropriateness value q_i; the returned normalized values are sorted to determine the optimal visualization approach. This behavior can be modified by simply reimplementing the evaluation function: A switch may change visualizations only if the instancing sequence of newly selected techniques finishes within a given time interval to maintain interactive feedback. Furthermore, additional criteria may be considered to support features like time-coherent visualizations to prevent frequent changes of the chosen visualization method or a preview rendering of all techniques for users who intend to manually select a method, based on preview images.

Context Modification via Context Switches

As context switches can freely implement the selection process even manipulations to the currently valid context can be made by the switch, when querying attached child techniques for appropriateness. Therefore, additional influence to the techniques' context evaluation and selection is possible, e.g., forcing aspects of the context to predefined values. Figure 4.5 shows a context switch that changes the available screen resolution of the original context to force a high-quality rendering of the visualization.

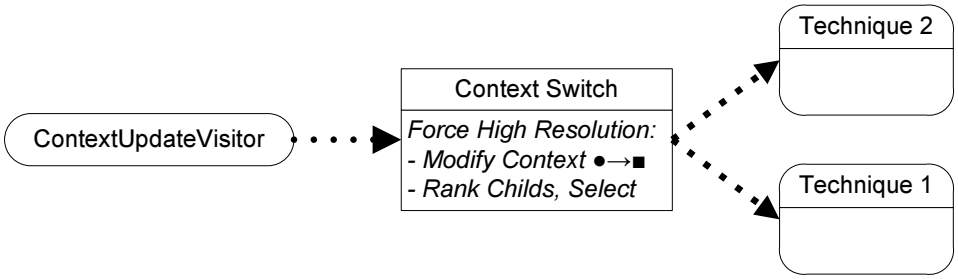


Figure 4.5: A context update with an advanced context switch that modifies the passed context to specific values.

4.3.4 Combined Rendering of multiple Visualization Techniques

No restrictions are placed on data requests that clients submit to the framework when requesting a visualization. This fundamental design choice frees applications from the requirement of submitting multiple queries if different data types have to be visualized concurrently. Often applications are unaware of data types or number of items a query might return; therefore, the framework should preferably take care of query results, even if multiple data types that require different visualizations are returned. In doing so, the framework can take advantage of combined vs. multiple queries: available visualization templates may consume multiple data types, e.g. air-flow direction and 3D geometry. If query results contain both types, a single visualization may be used to present both data, e.g. air-flow data on the surface of 3D geometry. In contrast, if multiple independent queries are used, a query on 3D geometry and a query on air-flow data, each dataset is visualized independently.

Even if multiple visualization techniques have to be used to present all data, their resulting images have to be combined in order to generate one single image. Simply adding the result of each technique to the final image can lead to artifacts, often due to incorrect depth sorting. Therefore, all visualization techniques that generate geometry—like stream lines, glyphs, etc.—are required to write correct depth values to the frame buffer. This way, visualization of opaque geometry can be intermixed with each other. For screen-space visualization methods, like the line integral convolution approach presented in Section 6.3, techniques have to either provide 3D proxy geometry, e.g. a billboard, or may utilize 3D geometry provided by the AWM to achieve data visualization on context-provided real-world geometry. Visualizations that make use of (multiple) transparent primitives—like the volume rendering approach introduced by B. Cabral et al. [32]—but do not spatially overlap with other visualizations, i.e. their bounding boxes do not intersect, are also correctly rendered. The underlying scene graph automatically detects transparent geometry and schedules its rendering after opaque objects in a back-to-front sorted manner based on bounding boxes. However, for multiple semi-transparent and spatially intersecting visualizations small blending artifacts are accepted due to partial incorrect depth sorting. Otherwise more advanced techniques are required, but even those are often limited to handle special cases e.g. interactive rendering of multiple intersecting volume datasets as proposed by F. Rößler et al. [170].

Similarly, common augmented reality applications require a mixture of camera images that present the view of the physical world and on top of it the computer generated information overlay, assuming the physical world is always further away than overlaid information. Unfortunately, this simple approach introduces artifacts whenever a physical object *should* occlude a virtual 3D object since for the physical world often no correct depth information is available. A widely applied approach to overcome this limitation is to pre-acquire the geometry of the physical environment into a 3D model and make use of it later to estimate the depth of the physical world by aligning this proxy model according the current context and query it for depth to decide if overlay information is occluded for each pixel of the image plane.

Context-Controlled Blending

In general, multiple visualization techniques may display information at the same image-space location; the above mentioned depth test ensures that only the nearest information is visible (assuming opaque primitives). This behavior, however, may be undesirable as important information of further away visualizations might be hidden.

Context-aware blending can be used, instead of simple overlaying, to render occluding visualizations semi-transparently revealing underlying information. Blending is foreseen in the rendering pipeline and efficiently supported by GPUs (see Section 3.1). The amount of transparency is preferably controlled based on the context: If a visualization technique generates image-space fragments then additional values are calculated to express the *importance* of these values, similar to importance fields defined by M. Zöckler et al. [213]. If following visualizations would occlude these fragments a blending is performed based on the stored importance value and the importance of the overwriting fragment.

The same approach is applicable for AR applications, i.e. visualizations on top of real-world video streams. This way, the occlusion of the physical world is minimized as only important aspects of the visualization, i.e. fragments with an importance above a given threshold, are shown. This concept is applied in the system presented in [57] and further detailed in Section 6.3.

4.4 Support for Mobile Devices

Mobility of users and mobile devices play an important role in context-aware systems as location is one of the most fundamental context aspects. Often context awareness solely makes sense for mobile systems; e.g. a mapping application that always shows a map of the surrounding area would simply present a static image for non-mobile users. But location and most other context aspects represent captured real-world data and therefore tend to change continuously which implies the requirement of processing context information at interactive or even real-time rates. For the domain of visualization this means that interactive techniques have to be developed to support context-aware visualization on mobile devices.

When mobile devices are supported, a variety of limitations of these devices has to be considered. Models ranging from highly capable PC-like hardware as tablet PCs, laptops, or ultra mobile PCs to light-weight mobile devices like smartphones or personal digital assistants (PDAs). All of them having different performance in computing or displaying data, as presented in Section 3.1.4. A major constraint on all mobile clients is still the limited network connectivity in terms of bandwidth and latency which is specifically addressed in the following optimizations.

The aforementioned system has been designed with the application of interactive mobile visualizations in mind, and the previously given definition of the framework for context-aware visualization already mentioned some important aspects to efficiently support mobile clients. The utilized client-server infrastructure even has inherent support for mobile devices: the caching and preprocessing server acts as a communication endpoint for heterogeneous client devices and optimizes the (wireless) transmitted data in a client-specific way.

4.4.1 Device-Specific Preprocessing

The final rendering of the visualization depends strongly on the client device capabilities and the available network connectivity. Several well-known approaches exist to enable interactive visualizations on smart mobile devices, namely: *Remote Rendering, Render Local, and Hybrid Rendering*, as detailed by J. Diepstraten [50] (note that his *alternative remote rendering method* is essentially a hybrid rendering approach).

- Remote Rendering: Makes use of a client-server infrastructure to calculate the final image of a visualization on a high-performance server. Afterwards, it is transmitted via network to a (mobile) client for display.
- Render Local: All required data and enough computation power is available on the client device so that the visualization is generated by the (mobile) client. Typically used for self-contained applications.
- Hybrid Rendering: Techniques that combine Remote Rendering with Render Local. The raw visualization data is preprocessed on the server, transmitted to the client, and further processed to generate the final visualization.

Most research, done for visualization on mobile devices, focuses on a single specific setup with known client devices and applications. In contrast, the proposed system for context-aware visualization supports diverse configurations and is able to dynamically select the most appropriate one, dependent on the available context information. Therefore, the proposed context-aware system can be used to deploy various visualization techniques to different clients and the framework automatically can configure the rendering technique. Again, the selection mechanism can depend on any per-client context information available to the system; some of the most common attributes are summarized in the following:

- Client-device capabilities: If the client hardware does not support functions needed by the selected display technique the framework reverts to Remote Rendering.
- Network bandwidth: If the available network bandwidth is too low for interactive image streams, the technique transmits the data to the client and initiates a (low-quality) Render Local implementation, adequate for the client.
- Storage requirements vs. network bandwidth: Large temporal datasets are inappropriate for an entire transmission. In cases where not enough network bandwidth is available, Hybrid Rendering is applied to continuously prepare parts of the raw visualization data and transmit it to the client which then calculates the final visualization.

- Application states: A fast adaptation on changing context information is useful to support low-quality locally-rendered visualizations during interaction and final high-quality rendering—generated on the server—when no interaction is performed for a predefined time period.

The selection of an appropriate rendering technique is based on the concepts described in Section 4.3.3 and integrated accordingly. As implementations of all visualization methods are based on the same basic scene graph API, all three rendering modes can efficiently be supported: For rendering locally on the client the required (sub-)scene graph is transmitted with its dependencies and executed on the client side to render the visualization utilizing OpenGL; if available, hardware acceleration is used. In situations where the final rendering has to be performed remotely, the server executes the (sub-)scene graph within a so-called camera node—an abstraction for hardware accelerated render-to-texture within the OSG API—resulting in an image of the final visualization using view parameters provided by the client's context. This image is then optionally compressed and transmitted to the client where it is displayed simply by extracting and copying into the frame buffer. Hybrid modes are integrated via specific visualization techniques of the *VDB* where computation is distributed between server and client. This way, a sever-based preprocessing is used to build an optimized data structure that is then transmitted to the client for calculation of the final visualization. Figure 4.6 shows the framework's specific behavior for heterogeneous client devices (compare Figure 4.1).

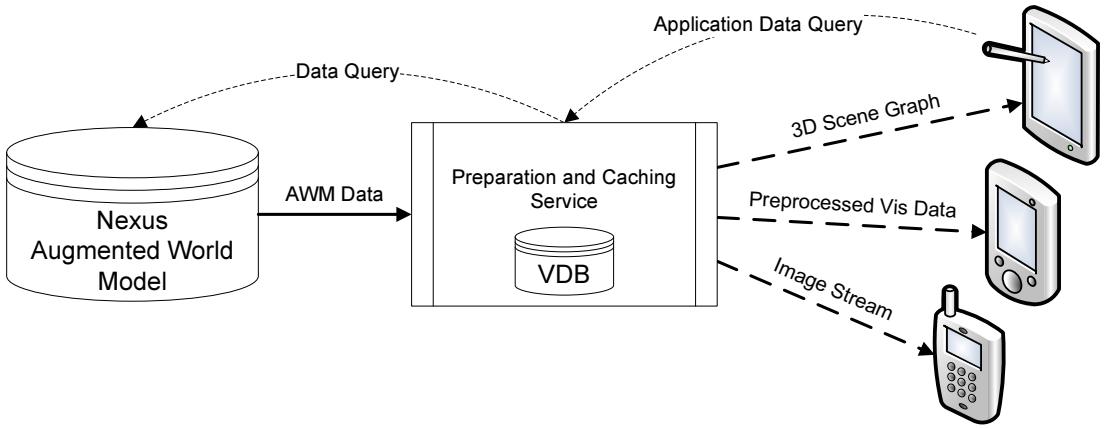


Figure 4.6: Framework support for heterogeneous client devices using different approaches for visualization on mobile devices.

4.4.2 Optimization of Network Communication

Independent of the utilized mode for rendering on the client, a network connection for data transmission is required. Modern mobile devices offer a variety of communication facilities: GPRS/UMTS using mobile phone networks, wireless LAN using local network infrastructure, or Bluetooth for low-range peer-to-peer communication. Each of these technologies offer different

network performance in different situations: WLAN offers a moderate bandwidth but requires access to nearby installed infrastructure, UMTS offers lower bandwidth but makes use of mobile phone networks, i.e. high availability. In general, performance of the network connection is often a limiting factor in terms of latency, bandwidth, or costs and has therefore to be optimized.

A simple approach to optimize the network communication is compression. However, if the semantics of transmitted data is unknown only lossless compression schemes for arbitrary data are applicable. Therefore, chunks of data that have to be transmitted could be compressed using, e.g., a ZIP encoding [44]. The disadvantage of this generic approach is that the computational requirements are often too high for interactive processing, especially on low-performance mobile client devices. If the structure of the data is known, more advanced techniques can be applied.

Compression of Image Streams

For rendering visualizations on low-performance client devices images and image streams offer good opportunities for optimization as even a lossy compression of the data is possible. Further, images and image formats are well researched and sophisticated algorithms are available to compress images.

Compression techniques to optimize remote rendering are examined by S. Stegmaier et al. in [188] and by J. Diepstraten in [50]. In principal, the goal is to balance computation requirements vs. size of transmitted data vs. image quality to achieve optimal performance. In contrast to per-image compression techniques, some work on remote rendering propose the application of compression technology developed for movie compression to utilize temporal coherence and thereby reduce the amount of data [147]. These techniques achieve a higher compression rate with the use of additional computation power for encoding and decoding the image stream.

Another approach for even more drastic reduction is to transmit images with a reduced resolution; however, this often leads to noticeable quality loss. On client devices these low-resolution images are then zoomed for display which often results in blurred images. However, the total amount of data that is communicated is significantly lower. An uncompressed 640x480 image requires about 900 kB whereas the size of a 320x240 image is about 225 kB (1/4). To alleviate the introduced quality reduction, a real-time visualization approach to minimize the mentioned blurring is discussed in Section 5.2.3.

Enhancements for the TCP Protocol Stack

For client devices that prefer to execute visualizations locally to improve interactivity, a compression of pure image streams is not beneficial. Data transmitted for such applications can only be compressed using a generic loss-less compression algorithm as their communication format is usually proprietary or even encrypted [140, 80]. Therefore, options to reduce the size of continuously transmitted data packages by these applications, which are in the order of tens of MBs, are very limited for the underlying network protocol stack.

Together with M. Scharf and C. Müller a modification of the Transmission Control Protocol (TCP) stack is used to optimized data transfer for the aforementioned class of applications, namely 3D web applications, as presented in [177]. The extension makes use of the communi-

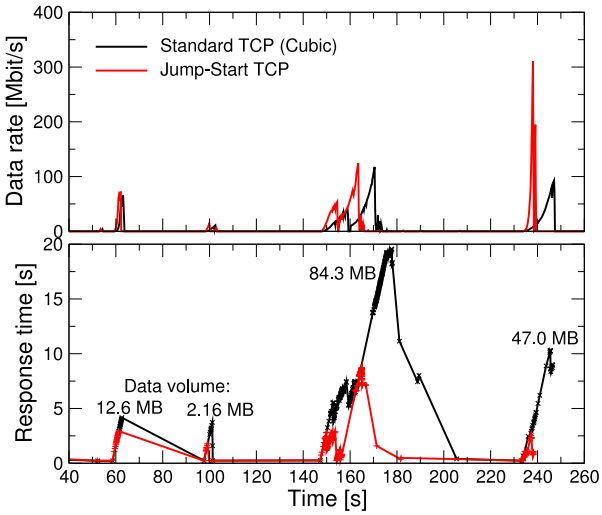


Figure 4.7: Data rate and response time measurement for standard and Jump-Start congestion control implementations.

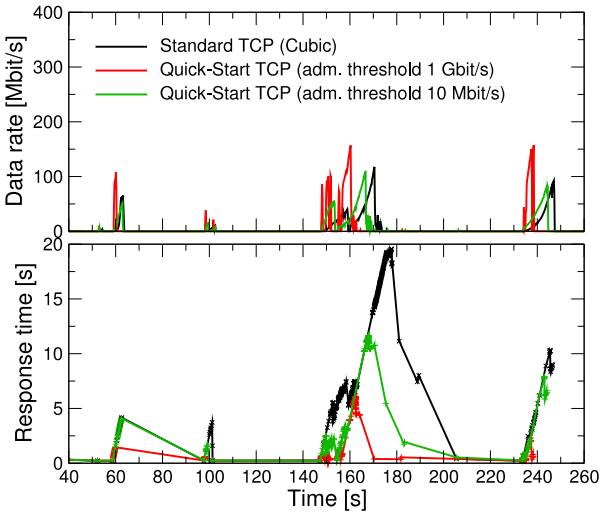


Figure 4.8: Data rate and response time measurement for standard and Quick-Start congestion control implementations on a 10 Mbit/s and 1 Gbit/s link.

cation pattern of such applications: users request data—through the selection of additional data layers or by changing the view point—and the server responds with several data packages of different size, most of them are nowadays in the range of tens of MBs.

Common TCP communication has a sub-optimal mechanism to support this kind of communication. The congestion control of TCP is implemented by a so-called slow start component; therefore, the transmission of each (large) data block starts using a moderate transmission rate which is increased while the transmission reaches the link's capacity. Replacing this slow start behavior with algorithms like Jump-Start [127] or Quick-Start [73]—replacement algorithms that try to use the entire available bandwidth instantaneously—a faster response time can be achieved for applications with this kind of communication schema. For interactive 3D applications that retrieve additional data on-demand, delays caused by data transmission times are strongly noticeable as users are forced to wait until all data is available to observe a complete visualization.

Using a 3D visualization prototype application that behaves similar to the mentioned geo-mapping examples an evaluation was performed to measure the benefits of TCP extensions. Figure 4.7 shows the results for a TCP modification according the Jump-Start [127] algorithm; Figure 4.8 depicts a TCP implementation utilizing the Quick-Start [73] enhancement.

The measurements were performed using a Linux-based implementation for the client and server component. The connecting network segment was a 1 Gbit/s Ethernet segment, but a round-trip-time (RTT) of 200 ms was introduced with NetEm [91] to simulate a realistic scenario were 3D data is retrieved from distributed servers around the world. During each measurement session a set of prerecorded user interactions were played back in order get reproducible results. As can be seen in the result Figures 4.7 and 4.8 if no user interaction or no additional data is required for a visualization on the client, no data is transferred over the communication link.

In both results the slow-start behavior can be seen for the standard (cubic) TCP congestion implementation. In fact, for some 3D data packets the download takes up to 20 s thereby the bandwidth is not fully exhausted. In Figure 4.7 timings for a Jump-Start configuration are included and clearly show the reduced delay for downloaded data. In best case scenarios even a 10 s reduction was observed providing a much better interactivity for end users.

Results in Figure 4.8 present both, the performance if Quick-Start is configured to utilize the entire available bandwidth (1 Gbit/s admission threshold) and if the allowed bandwidth load is limited to 10 Mbit/s. When utilizing the entire available bandwidth, Quick-Start achieves similar acceleration as observed for Jump-Start. Interestingly, measurements with a limited 10 Mbit/s utilization still showed a performance improvement due to the faster bandwidth utilization.

In conclusion, the experiments showed that for specific communication patterns existing congestion-control implementations of TCP are sub-optimal. Improved performance of context-rich geospatial applications can be achieved not only by optimizing the applications' communication but also by enhanced extensions of the underlying network communication protocols. However, the presented measurements target a scenario of large-scale distributed context-providers where a round-trip-time of 200 ms is realistic; for local area network configurations that include server and client component the RTT is much smaller resulting in lower improvements (already for 50 ms RTT max. $\sim$1 s improvement).

4.5 Context-Aware Interaction

The focus of this thesis is clearly on techniques to support context-aware visualization. However, as no restrictions on context information are made, arbitrary context aspects are considered that may influence the results of visualizations including user interaction, which can also be interpreted as context information. In contrast, context data can be used for user interaction making common applications context aware, as considered in numerous research projects.

Alternative input interfaces to navigate through menus on a hand-held device are presented by J. Rekimoto [167]. J.F. Bartlett shows an alternative input technique where the orientation of a hand-held device is used to scroll and navigate within an electronic photo album [13]. A wider interpretation of context is proposed by A. Schmidt et al. in their publication: *"There is more to Context than Location"* where context information includes environmental lighting condition and device orientation [180]. But not only research projects work on context-aware functionality, the recently introduced Apple iPhone [6] integrates an orientation sensor used for sensing, e.g., the device orientation and can automatically switch from portrait to landscape mode as proposed by A. Schmidt et al. in [180].

As the proposed framework already makes use context information and context-aware visualization will benefit from intuitive user-friendly interaction possibilities—especially on mobile devices—the potential of a generic orientation-aware user interface is evaluated. Together with S. Stegmaier and D. Weiskopf various usage scenarios were examined that may benefit from an orientation-based user interface [60]. A custom prototype based on consumer-level PC hardware equipped with an inertial orientation sensor was built (see Figure 4.9) for evaluation of various mappings of orientation according to the approaches explained in Section 2.2.3. With the use of

an orientation sensor the system is able to provide an additional means to interact with the device in an intuitive way.

4.5.1 Generic and Custom Orientation-Based User Interface

The evaluated scenarios include a generic orientation driver implementation that can be used for any existing application, an augmented reality explorer, and a computer game. Different mapping concepts are applied—as defined in Section 2.2.3—to implement the orientation-aware behavior. To find adequate mappings and parameters for the examined applications, several user interviews were evaluated during the implementation of the prototype.

Figure 4.9: Tablet PC with a mounted inertial sensor.

Generic Orientation Driver

An evaluation of a generic driver for orientation-aware user interaction—targeted for a hardware setup as seen in Figure 4.9—is motivated by the option to integrate user interface techniques within the context-aware visualization framework. The generic driver application supports both operation modes a clutch mode and a lock mode (see Section 2.2.3) to activate the orientation awareness which is signaled by an animated tray icon. In the default configuration, orientation is mapped to scrollbars using a continuous mapping to allow arbitrary scrolling speeds [13, 89]. Therefore, any application which uses scrollbars benefits from the new input technique. A schematic illustration of the driver functionality is shown in Figure 4.10.

For applications not making use of scrollbars, input via tilt gestures is exploited by sending key-press events to the application. The generic driver implementation allows a per-application discrete key-mapping configuration. When an orientation change is recognized, the system searches the setup for the current window in focus. If no special configuration is found, the

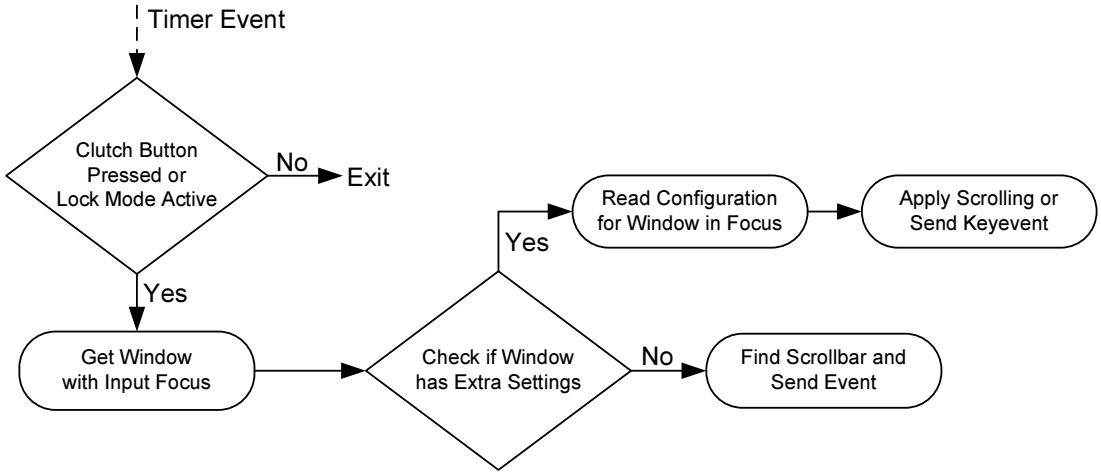


Figure 4.10: Algorithm to map orientation changes to scrolling or key-press events.

default behavior—mapping orientation to the scrollbars—is applied otherwise the configuration is read and the specified events are triggered, e.g. key-press macros. This feature is used to, e.g., implement page turn-over in the *Microsoft Reader* ebook reader by tilting the device left or right. As ebook readers organize the content layout in pages without scrollbars, just like real books, mapping orientation to scrollbars would not have the desired result a user would expect. This allows users to configure a wide range of applications to be operated via tilt operations; however, it also reveals the need of additional context information to automatically map orientation to an adequate functionality. Such additional context-information might be provided by an underlying context framework like Nexus.

Considering mobile client devices, an often desired application is a full-featured internet browser application. But common user interfaces of browser are not suitable for smart mobile devices: They make use of scrollbars, tabbed windows, popup windows, hyperlinks at arbitrary locations within a web page, etc. Therefore, internet browsing is used to evaluate the effectiveness of a generic driver to control sophisticated applications with which users are familiar. To navigate within HTTP documents with tilt operations the problem was encountered that users want to scroll the document and additionally want to step through the included links. Naturally, both operations should be mapped to the same up or down tilt gesture which is, in general, not possible. An adequate solution is to scroll and step through the links simultaneously: When users tilt the device the information display steps to the next link if and only if the link is contained in the currently visible area of the document. In all other cases the page is scrolled. This behavior can be further refined so that the system only steps to the next link if it is shown in a user defined area of the screen. The image sequence in Figure 4.11 shows successive tilt operations, the greenish area defines the region in which a link must be visible in order to be selected.

Additionally, left tilting is discretely mapped to step back to the previous page of the displayed information and with right tilt operations it is possible to select links within the HTTP document. An open question for browser interaction is still an effective method for entering of URL addresses without using a keyboard.

Figure 4.11: Tilt operations result in successive scroll and link-step actions. Initially, the first link is marked. After a tilt gesture the next link is selected. Tilting again scrolls since the following link is out of the greenish *link area*. A further tilt scrolls and additionally marks the last link.

The generic driver prototype can also detect the orientation relative to the 1G gravity vector which is used to implement an automatic screen configuration from landscape to portrait orientation and vice versa [180]. A threshold prevents screen reconfigurations if no significant orientation change in either direction is recognized, e.g. if the device is lying on a table. If the driver detects the gravity in inverse direction, i.e. the device is facing the ground, it is assumed that users do not operate the device in this state and the system enters a suspend mode.

Augmented Reality Explorer

Common augmented reality systems make use of location and orientation in order to augment real-world views with virtual information. While sensing the orientation is handled with inertial sensors, the absolute location cannot be determined easily and often requires costly additional hardware as discussed in Section 2.2.1. Using only orientation information, still limited augmented reality applications are possible under following assumptions: users explore only single objects and maintain a fixed distance to them. This way, orientation information is enough to roughly estimate the position of the display relative to the object. The prototype implementation displays a virtual object at the same location and orientation as the physical object. Users can explore the object from different views without learning how to navigate, just by moving the

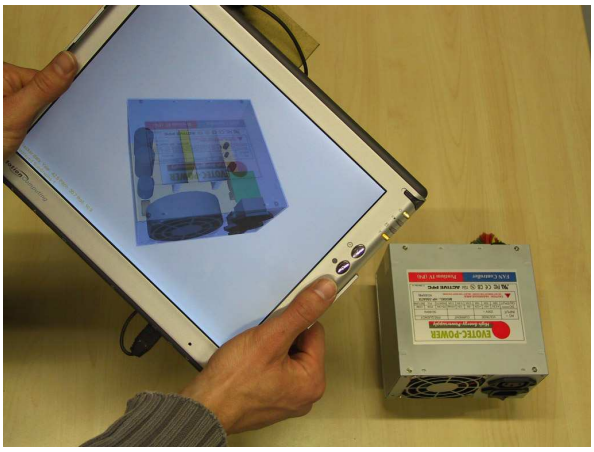


Figure 4.12: AR-Explorer to explore objects by orienting the display and switching irrelevant parts transparent.

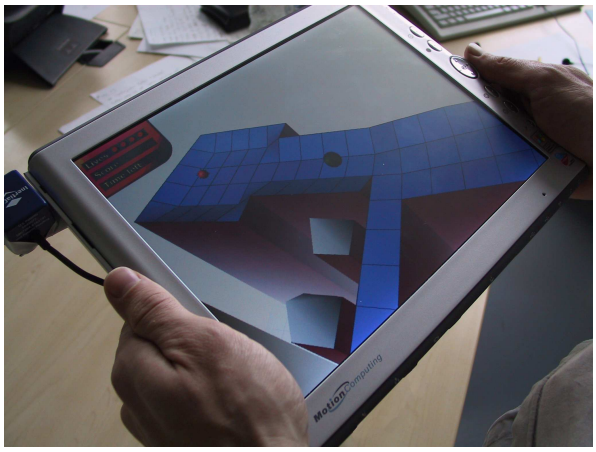


Figure 4.13: Playing *Trackballs* (a Marble Madness clone) via tilting the device.

display around the object as shown in Figure 4.12, similar to an AR window as often used in medical applications [181].

Two different modes of operation are supported: Orientation is permanently used to synchronize the view of the physical and rendered virtual object as seen in Figure 4.12. If the clutch button is pressed or the lock mode is activated, orientation is mapped to user interaction functions that allow users to select different parts of the object with left or right tilt gestures and to modify them by tilting the device up or down. For selection of parts a continuous mapping is applied to allow operators a precise (*slow*) navigation with slight tilt operations and fast stepping though the parts using more extreme tilt gestures. To modify the selected part of the virtual object a partially constant mapping is used to apply four different functions modifying the selected part.

Gaming Entertainment

Sensing the orientation of the device also gives rise to new applications and interaction techniques. The recent success of Nintendo's Wii gaming console [146] clearly shows the demand of new innovative interfaces. In entertainment, orientation sensing is used for games of many kinds as, e.g., racing games or simple dexterity-puzzle-like games. For evaluation of orientation-based interfaces for games the open-source game *Trackballs* [28]—a Marble Madness clone—was adapted so that the device's orientation is used to steer the ball as illustrated in Figure 4.13.

4.5.2 User Evaluation of Interaction Concepts

A short user poll was accomplished to evaluate the prototype applications. Users were given a quick introduction to the orientation-aware system and its basic techniques to map orientation to user interaction. Thereafter, users operated the example applications and had to rank the user interfaces on several aspects. The scale of the rating for each aspect ranked from minus two up

to plus two. Seven users participated in our study, for all of them the concept of operating a user interface by changes in orientation was totally new. User feedback on how comfortable the application can be operated is expressed in *Handling*. The *Advantage* item ranks the benefit of orientation input versus common input techniques. Whether the new input technique provides enough precision is expressed in *Precision*. Finally, in *Intuitively Usable* users had to rate how intuitive the interface can be operated. Results are summarized in Table 4.1.

Table 4.1: User feedback on prototypes.

	AR Explorer View Orientation	AR Explorer Part Selection	Internet Browsing	Generic Driver Scrolling	Gaming Entertainment
Handling	+	+	−	+	+
Advantage	(++)	+	−	0	+
Precision	0	+	0	+	++
Intuitively Usable	++	++	−	++	++

Some characteristics of the orientation-aware user interface could be seen in all interactions that were performed. Due to the device configuration the clutch mode can hardly be operated in portrait orientation as the button can no longer be used ergonomically (see Figure 4.9). But this problem is even more severe if common user interface techniques are used where more buttons have to be operated to execute the same interactions. B.L. Harrison et al. addressed this problem in [89].

AR Explorer View Orientation To evaluate the view manipulation of the AR Explorer the users had to explore various parts of the examination object. All users intuitively used the tablet PC as a window and could easily perform the task. Two users claimed the weight of the device restricts the handling. The advantage of using orientation information to determine the view direction versus defining it manually, e.g., via cursor keys, is huge. Nevertheless, compared to other AR window devices there are disadvantages, especially in terms of precision and drift of the inertial sensor.

AR Explorer Part Selection For selecting subparts of the exploration objects most users chose the clutch mode as they sometimes walked around the object to get a better view of the currently selected part and, therefore, had to deactivate user interaction. Four users preferred an ordinary user interface using left/right cursor keys to select parts.

Internet Browsing Users were able to scroll the web site intuitively. However, many users had problems selecting/following links and afterwards returning to previous pages. All users needed further explanation to understand the concept. Five users claimed that the interaction method is non-intuitive and too complicated to be operated easily.

Generic Driver Scrolling For examination of user interaction using the generic orientation-awareness driver a scenario when reading a long (approx. 8 screens) text was analyzed. Most users used the lock mode for reading, setting the reference plane which defines the default orientation for no interaction (see Section 2.2.3) in a way that the window content scrolled at a speed corresponding to their reading speed. Two participants used the clutch mode. They scrolled step-by-step each time by pressing the clutch button, scrolling, and releasing the button.

Gaming Entertainment All users naturally interacted with the device and tried to keep the ball on its track. Participants of our study were greatly amused by its simple and intuitive handling and voted for significant improvement in interaction versus ordinary keyboard interfaces.

Application of Orientation Awareness

Several scenarios are examined to evaluate the universal application of orientation for user interaction. The performed user poll to evaluate the proposed methods showed that some applications benefit from a context-aware user interface, especially applications that naturally map orientation to orientation-like behavior as AR scenarios or gaming applications. Nevertheless, the evaluation also showed that for some cases the presented user interaction techniques are inadequate and additional concepts have to be found. Therefore, a tight integration of an orientation-aware user interface within the framework for context-aware visualization was not further considered. Nevertheless, some developed prototypes still map orientation to user interaction behavior as shown in Chapter 6.

4.6 Concluding Remarks

Development and deployment of context-aware visualizations is greatly simplified with the proposed framework. Its flexibility to integrate new techniques, support for GPU-based techniques, and focus on mobile devices makes it a generic framework applicable to various scenarios. As the entire system is based on the underlying open-source scene graph OSG [155], existing basic visualizations and even existing end-user applications can easily be adapted to make use of the framework. By integrating an underlying context-aware data management platform, like Nexus, a huge federated data store containing heterogeneous context-based information becomes available as a data source to generate visualizations for client applications. Complex interactive visualizations composed from multiple data sources of different formats are supported by the framework and available on various device types.

Although the proposed framework provides a solid basis for context-aware visualization applications, there are still open issues that have to be addressed in future research to further improve the framework. The current design allows each visualization template to freely implement its appropriateness weighting function to rank itself. This flexible approach has the drawback that developers have to design a scale to compare techniques versus each other to implement weighting functions that return values that match to other technique's weighting function. Furthermore, if one visualization is capable to fully satisfy one context aspect but completely ignores another

and a second visualization exists which has the opposite characteristics then the framework cannot decide which visualization is best. In certain circumstances even a composition of both techniques might be the best option. To address such undecidable cases, a more sophisticated reasoning system could be integrated into the framework and visualization templates should negotiate which context aspects are handled and which have to be considered by further techniques. The interesting approach shown by J. Mackinlay [133] to automatically generate 2D visualization for relational data might serve as a basis to evaluate and combine multiple visualization techniques to present all aspects of a single dataset. Also ontologies will play an increasingly important role to build intelligent systems to reason and interpret context-aware information.

5 Context-Aware Visualization

In this chapter visualization techniques are detailed that present raw input data according to the current situation, estimated based on context information. Actual prototypes are presented—based on the framework proposed in the previous chapter—as exemplary techniques to demonstrate the advantage of context-aware behavior and motivate future applications.

Specifically, sophisticated interactive presentations are examined that show the advantage of the framework's support for graphics-hardware based methods and the efficient processing even of fast-changing context aspects. In addition to an interactive visualization, the importance of context information is discussed: Already focus&context techniques identify the value of context data and use it to visually present information that is related—usually the spatial neighborhood—to the examined (focused) data; context-aware visualization extends this approach and utilizes context information to actively influence the presentation of data. However, presentation of data combined with context information is still an important aspect in context-aware visualizations.

When using context to control or configure visualization results, each part of the visualization pipeline may be influenced (see Section 3.3). The last section of the visualization pipeline is represented by the rendering pipeline which is responsible for generating the actual pixels of the final image. Often, the rendering-pipeline stage of an entire visualization can be used to adapt visualizations to match, e.g., different output devices, independent of the underlying visualization method. The remaining parts of the visualization pipeline are defined by the applied visualization techniques, which are selected based on context information out of possibly multiple alternative approaches. Finally, the mechanism of context-aware configuration and fine-grained adjustment of visualization techniques is shown using concrete examples and implementations.

5.1 Combined Presentation of Context and Data

Visualization is the task to visually represent (complex) data; context-aware visualization further improves this process by adjusting parameters based on context information. However, dependent of the considered context aspects such an automatic adaptation might be counterintuitive to users since the underlying context is a priori unknown. Major context aspects are location and orientation—i.e. placement within the environment—so that context-aware visualizations that are influenced by these aspects often present the surrounding physical environment to convey the relation to its context and make the automatic adaptation process more intuitive to users. In

contrast, visualizations might also be embedded within a different, even a non-physical context, i.e. a virtual or artificial context, to interpret or analyze data within different contexts.

When presenting context and data, sometimes the context information is equally important as the visualization of the data itself. In this case, the suppression of relevant context information due to data visualization has to be minimized. Therefore, visualization has to be integrated with minimal interference of the presented context, which is addressed using embedded visualization (see Subsection 5.1.3).

Furthermore, presenting context information in addition to specifically considered data is especially problematic for small-screen devices, since visualization is already a dense optimized representation of information often without much screen space left to display context data. Therefore, techniques to increase the available display area are popular: Level-of-detail approaches and zooming allow for freely adjusting the ratio of data visualization versus context presentation. Such techniques are even beneficial for common desktop applications, as not only augmented reality-like systems make use of the principal concept to present data within its context; often data/information is more easy to interpret and understand if related, surrounding data is also shown.

Focus&Context in Visualization

Approaches that present both, details of the explicitly considered data, the *focused* data, and (parts of) the related data, the data's *context*, are termed *focus&context* visualization. Promising methods are developed in the field of information visualization where often huge amounts of information and relations have to be illustrated. A prominent example is the work on fisheye views by G.W. Furnas, which is based on the characteristics of a physical fisheye lens [76]. Important aspects in the center are enlarged; whereas off-center regions are scaled down to generate single images that show details and an overview concurrently.

Application of focus&context techniques in scientific visualization, among others, is proposed by H. Hauser. In [90] several well-known visualization techniques are enhanced with focus&context functionality to improve their usefulness. The presented benefits, when context of data visualizations are presented, further demonstrate the requirement to support combined visualizations for data and its related context.

5.1.1 Real-World Context in Augmented Reality

Augmented reality can be interpreted as a simple realization of context-aware visualization, as already defined in Section 3.4. Figures 5.3b, c show two primitive examples of augmented reality visualizations: A scalar information is presented via color coding and overlaid on top of an image that represents the physical context of the information. Thus, AR is a focus&context visualization as data and related context information is presented and is further a context-aware visualization as the presentation is adapted or synchronized to the real-world view that captures the environment. When considering Figure 5.3c one can imagine that without the contextual information of the underlying image, the augmented information cannot easily be interpreted by users as the spatial relation of the data to the examined scenario is not shown.

The development of applications even with limited context-aware behavior, as augmented reality approaches, is already challenging. AR tries to spatially align information within a physical context at high accuracy, which implies that location and orientation context information have to be available in high precision. Available infrastructure-based tracking systems are able to provide this information but limit the applicability to specific applications as shown, e.g., in [208].

In contrast, the definition of context-aware visualization widens the concept of AR applications so novel approaches will emerge that generate further use cases. An example are orientation-aware applications—versus location- and orientation-aware augmented reality—that make use of simpler, infrastructure-less hardware based on inertial sensors to measure orientation changes, and users are supported in mentally mapping the orientation-aware visualization to corresponding physical locations. In Chapter 6 and in [60, 207] several prototypes are presented that make use of this strategy. In addition to considering different context information for context-aware behavior, also the presented physical context of AR-like visualization can be changed to enable further fields of application.

5.1.2 Advantage of Virtual versus Physical Context

Focus&context methods aim to provide a deeper insight into the examined dataset by embedding the visualization within its related data. In augmented reality the contextual information is provided by a view to the real-world, therefore the context matches the physical context. In a research project together with O. Siemoneit the impact and benefit of real versus virtual (artificial) context information is examined [59].

The reality-virtuality continuum, detailed in Section 3.4, defines possible setups to combine virtual and real information. Research projects often focus on a specific configuration: augmented reality [195] or virtual reality [69]. Scenarios that utilize the entire continuum are not well researched; in contrast, an application that benefits from multiple different configurations of mixed reality are presented and evaluated in the following and published in [59].

An Application using a variety of Mixed Reality Traits

A scenario where mixed reality can efficiently be combined with virtual reality is, e.g., a customized production process. It can be observed that manufacturing is nowadays often adapted from mass production to individually configured products. This tendency results in a modified production environment; workers have to decide which part and configuration has to be chosen to build up certain customized machines. A conventional augmented or virtual reality system cannot fully support a worker in these tasks. Both technologies can only cover some aspects; whereas the proposed MR/VR system combines advantages of both systems via the ability to seamlessly change the amount of real versus virtual context that is presented.

First, a virtual reality preview of the deployment environment helps workers to choose the basic product and its configuration for this individual installation. While assembling the product a worker may choose—based on the difficulty of the task or his experience—the level of assembly instructions. Several levels of mixed reality can be supported within the range from no instruction (reality) to virtual reality. The decision on one of the different product configurations that

is most suitable can easily be made in an augmented virtuality view: The product is virtually installed in its deployment site in the current state of assembly. The worker can orient and position the product and view it from different points of view while the virtual installation environment is always displayed in the corresponding orientation. This way, an intuitive interaction supports the worker to visually choose the best available configuration option; e.g. connectors have to be installed on the most adequate position for accessing, installation, service, etc. Figure 5.1 depicts a worker using the proposed prototype to assemble a control unit with a power supply.

Figure 5.1: A worker assembling a power-supply module into a control unit using the presented prototype.

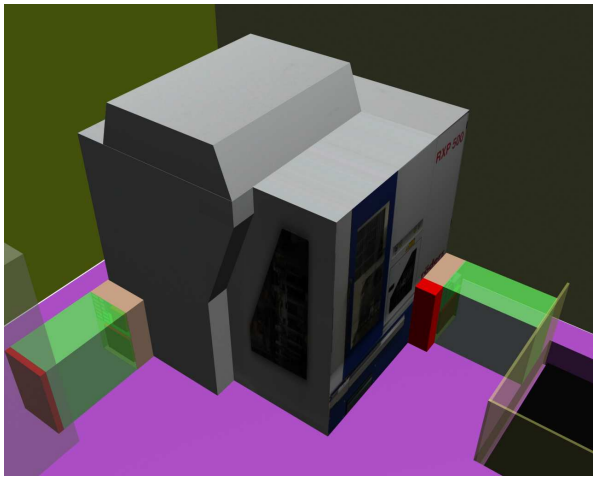


Figure 5.2: Virtual reality preview of the customer's manufacturing site with a visualization of possible installation options and maintenance areas.

For the implementation of a prototype that supports transitions between different reality stages there additional hardware requirements versus common, single-stage augmented or virtual reality applications. The display device—the prototype makes use of an adjustable optical-see-through head-mounted display[1]—must support different operation modes: see-through, non-see-through, and intermediate modes. As common for AR or VR applications high demands on interactivity are placed that are addressed in the prototype by utilized graphics hardware to achieve low-latency visualization based on the OpenGL-based scene graph OSG [155]. The prototype implementation supports the following operation modes:

Tutorial Video For training the personal to perform a certain task it is often more efficient to show a prerecorded video clip of real, already trained workers accomplishing the same task. (Note that this mode was not used during the user evaluation.)

Virtual Reality Preview This mode displays a VR scene on the HMD and users are able to navigate within the scene. In addition to the single 3D model of a machine, the entire installation

[1] Refurbished and modified Sony Glasstron PLM-S700 from Cybermind Interactive: http://www.cybermind.nl

environment is shown to provide users with hints how the machine is placed in the customer's manufacturing site. This way, a worker, who has to configure additional units to be installed on the machine, can easily see collision problems and can select the best available installation option for each customer. Figure 5.2 illustrates a milling machine with possible placements of the control units (orange). The attached maintenance area is visualized in green, if no collision is detected, and in red, if the area is obstructed by other installed parts of the manufacturing site (Figure 5.2 left).

After choosing the installation position of the attachment, using the VR preview, the prototype switches to an augmented reality manual for the configuration of the attachment part.

Augmented Reality Manual The internal configuration of attachment parts, e.g. a control unit, is often dependent on the position where the part is mounted on the machine. The system automatically augments the physical control unit with virtual parts that have to be installed at the correct position. Workers can then easily place real objects at exactly the same position as the virtual ones; without the need to read technical plans or installation manuals. The prototype thereby shows (animated) virtual installation parts, mounting points, screws to be installed, arrows to mark important positions, and text labels for short textual information.

Augmented Virtuality Review For the final inspection of the control unit and its inner configuration an augmented virtuality view of the physical control unit is supported, combined with either a virtual machine or even the entire customer's manufacturing site. Thereby, the visualization of the virtual environment is automatically adapted according to the user's view to the physical unit that has to be installed. This mode is very useful for the final control of the assembled unit in order to recheck location and inner configuration of any collisions or other hindrances.

Results of an Explorative User Test

For efficiency and effectiveness evaluation of the prototype a small explorative user test was accomplished. Ten persons divided into two groups participated individually in the test. The task for both groups was to correctly assemble a control unit casing, which is an exterior extension of a customer's CNC machine. The first group was asked to assemble the casing in the traditional way, without the usage of the proposed prototype. Therefore, a location plan of the customer's factory floor and detailed engineering drawings of the machine in front, top, and lateral view were provided. Based on this information, the participants had to decide, with respect to certain maintenance and clearance distances, on which side of the machine the casing should be attached. Furthermore, dependent on the mounting position the participants had to determine inner setting and arrangement of the casing.

In contrast, the second group was asked to use the developed prototype for the same task, utilizing all features mentioned in the previous section. The user test included qualitative and quantitative methods. As quantitative performance measures the amount of correct task completion and the total assembly time were used. But most information—as appropriate for an explo-

rative study—was gathered through qualitative methods like task surveillance, non-standardized interviews, and a think-aloud usability test.

Results of the usability test show that the difference between the two groups is significant: Four of the five participants with mixed and virtual reality support were faster than the participants using the traditional method. The average assembly time was roughly two minutes less, at a task completion rate of 100% in both groups. The discrepancy of fastest assembly times of 3 minutes 46 seconds in the group using the traditional method, in contrast to 1 minute 46 seconds of the group using MR support, indicates major problems in reading technical drawings; raising the question whether the example was representative enough or not. The qualitative data collection further confirmed the difficulties in reading floor-plans and drawings.

Overall, the results of this first explorative user test indicate potential benefits of MR/VR applications in flexible manufacturing environments. The implemented prototype demonstrates how multiple reality stages can be combined for practical applications leading to lower assembly times and lower error rates. Specifically, the advantage of providing context information, i.e. a visualization of the corresponding environment (real or artificial/virtual), help users to interpret situations and make decisions.

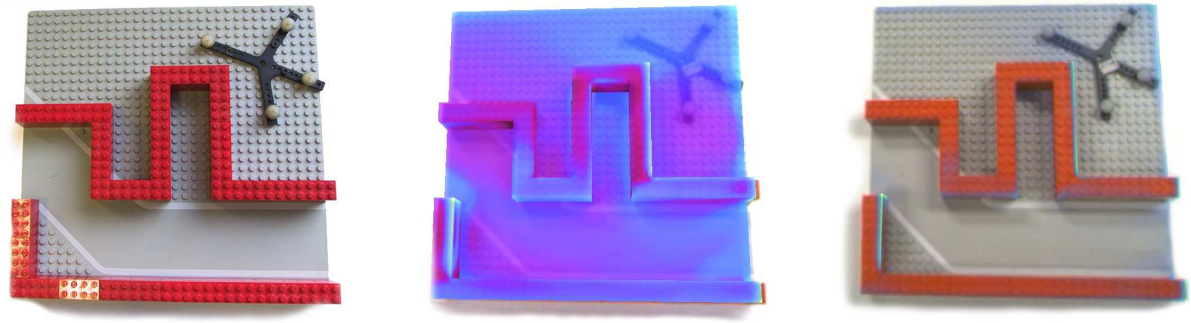
5.1.3 Embedded Visualization

In previous sections the advantage of presenting related context information in addition to the considered (focused) data is discussed. Thereby, the problem arises that presentation of contextual information and visualization of the data itself have to divide or share the available screen space. In focus&context approaches often a degree of interest function is evaluated that classifies the data into important and unimportant regions, but such classifiers are not always available.

A related approach tries to estimate the importance of the visualization within a region—this could be a screen-space tile or simply a single fragment—if the importance is low the corresponding screen space is used to convey the context; highly important regions are used to present the visualization. This way, the visualization is reduced to important aspects and thereby minimizes the occlusion of context information. Especially AR-like systems benefit from this as in general no information is available that indicates any kind of importance within context, which is therefore considered everywhere equally important. Therefore, minimizing the overdraw caused by the augmentations when designing AR applications is often advantageous so that more display area can be used to present the context, but still some amount of screen space is required to display the information. Optimizing this balance of screen space used for information visualization versus context presentation is, e.g., addressed by B. Bell et al. [20]. For annotation of physical objects the presented work avoids occlusion by optimizing the placement; however, this work is targeted for information icons and cannot easily be used for visualization tasks.

In general, for combined visualization a minimal interference with the also displayed (physical) context information is advantageous. If additional information about the context is known—especially regions that allow overdraw without hiding important aspects—then these regions may be used to present augmentations, as already shown for text labels by B. Bell et al. [20]. Further, if entire surfaces and their alignment in the physical world are known and they do not convey important context information, e.g. the planar surface of physical object or simply a solid-colored

wall of a house, then this space can be reused by visualization for augmenting the view with additional information about the underlying physical surface. The visualization is thereby smoothly embedded within the surrounding context and is therefore termed *embedded* visualization (also: visualization overlay). In Figure 5.3 two examples of embedded visualizations are presented: The original non-augmented scenario is shown in a); b) shows a visualization that utilizes all physical surfaces of the scenario to convey a scalar value related to the underlying surface areas. Note that the geometry of the scenario can still be observed as the physical surfaces are just semi-transparently painted with information. In c) the same approach as used in b) was applied, but only important values are shown, which are in this case at the edges of the geometry. Note that this basic approach of color coding is popular as it allows very fast visualizations even on low-end devices and can easily be implemented with rudimentary functionality of graphic processing units.



a) b) c)

Figure 5.3: Embedded visualization by overlaying physical surfaces with virtual information. The physical scenario a) is semi-transparently augmented with scalar information leaving the underlying scenario still visible, as shown in b). Even less aspects are hidden in the importance-driven approach depicted in c).

As mentioned, in order to realize an embedded visualization knowledge about the geometry of the scenario is required as the basic idea is to overlay or partially replace real-world surfaces with virtual ones. However, the same data—namely a full 3D model or at least definitions of the to-be-overlaid surfaces—are required to correctly handle occlusions of visualization geometry with physical objects as detailed in Section 4.3.4. Examples that make use of embedded visualizations to implement context-aware visualization are presented in [57] and also in Chapter 6.

5.1.4 Zoomable User Interfaces

Independent of the chosen approach for a combined focus&context visualization, developers and users will almost instantaneously find situations where not enough of the context is shown or details of the considered data are not visible. Ideally, the amount of shown focus or context data should be freely configurable. Zooming seems to be a preferred concept to interact with

focus&context visualizations: *zoom in* increases the details that are shown; *zoom out* allows to present more related data.

This generic concept can be applied to various applications and is not limited to visualization tasks. Therefore, the most basic software users interact with is used to examine the benefits when interacting with focus&context presentations using zooming: the user interface of the operating system. Previous work on zoomable user interfaces published in Pad respectively Pad++ [158, 18]; B.B. Bederson used the experience obtained in the development of these projects to build the toolkits Jazz [17] and Piccolo [16]. J. Raskin started the open-source project Archy (also known as The Humane Environment) [164] that implements metaphors for navigation in and usage of zoomable user interfaces. Further, there are several studies that compare the concept of zoomable user interfaces with common methods of user interfaces [96, 85].

Zoomable user interfaces support the realization of focus&context methods as they allow users to freely choose the amount of context they want to see. This is achieved by enhancing common graphical user interfaces (GUIs) with different zooming functionality. Results of a general approach to implement a zooming desktop interface is presented in cooperation with M. Rotard and R.v. Putten [172]. The proposed evaluation prototype makes use of scalable vector graphics (SVG) as an underlying technology, to support mathematical description of shapes and positions and offer an operating system/display device independent implementation. The novel interface paradigm solves the problem of the limited screen real estate and enables users to have windows, documents, and folders at any location on the desktop in a preferred presentation size. Humans tend to remember landmarks and relative positions, which is utilized by zoomable user interfaces where information is accessed on the desktop by zooming and panning to a specific view. Particularly with regard to small-screen mobile devices, zoomable user interfaces are an advantage for users handling several, possibly complex applications concurrently.

In such systems, document icons on the desktop present a preview of the document that will increase in detail—showing its content—when the user zooms in. The displayed size of a content—adjusted by the user via zooming—can be interpreted as context information indicating the user's intention. Exploiting this, a user interface can transparently switch applications while zooming in on, e.g., an image, which might be embedded in a text document. At a defined zoom level, the image will directly be editable within the primary document—using functionality and the interface of a different application—whereby the contextual information of surrounding document will still be visible. A zooming user interface can thus support users in handling the still increasing number of different applications that are used simultaneously (e.g. for writing this thesis).

Furthermore, many applications already provide different kinds of zooming, e.g. image editors, text editors, or document viewers. This shows that zooming is an often applied functionality, but each application makes use of it in a slightly different manner. For that purpose, users have to learn and understand the different kinds of zooming and their corresponding interaction technique to make use of them. To minimize this effort, in [172] paradigms are proposed to further improve zoomable user interfaces and unify the zooming techniques of different applications to a general concept for zooming named: three-level zooming. Furthermore, application level-of-detail zooming is presented to transparently switch to applications, which are used to generate specific content within contextual documents or other applications.

Three-Level Zooming

The goal of three-level zooming is to support zooming within the underlying window management in a way that applications do not need to support zooming, as they automatically make use of the zooming functionality provided by the operating system's user interface. Three semantically different levels for zooming have been identified for a zoomable user interface, which are illustrated in Figure 5.4 and detailed in the following.

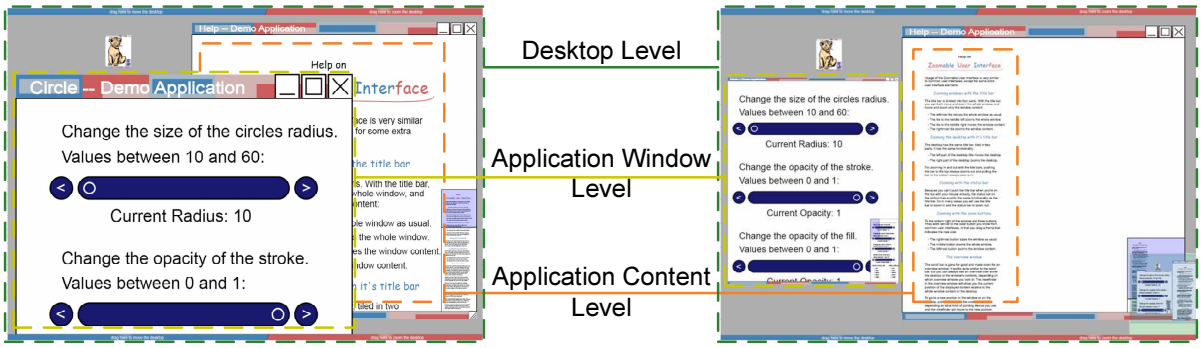


Figure 5.4: Example illustrations of the proposed three levels for zooming: desktop level, application window level, and application content level.

Desktop Level Zooming the entire desktop environment is used to choose the level of detail of the entire working space. In general, this basic level is supported by nearly all proposed zoomable user interfaces. Through this functionality users can easily get an overview of the entire desktop by zooming out in order to view all nearby windows simultaneously. At this level the windows may only show a rough approximation of their content; however, still enough for a user to recognize all active applications and their corresponding windows. To get more detail of a specific area, the user can simply zoom into the area of interest. Thereby, the context information of all opened windows diminishes continuously until only a single application window is shown on the entire screen.

Application Window Level Resizing the entire application window including menu bar, short-cut icons, and content. Often, applications provide a user interface via menus, short-cut icons, buttons, etc. The more powerful an application is, the more advanced and complicated its user interface gets. In contrast, simple applications with a limited functionality mostly provide only a simple user interface with, e.g., only a single button. On today's windowing systems users are forced to rely on application programmers to choose an adequate size for interaction elements. On a zooming user interface the display size, on which the application is operated, has a much higher variety than for fixed-size user interfaces. Although this fact makes it hard for programmers to choose an adequate user interface design, inadequate interface sizes are much less severe as users can easily adjust it according their needs. In addition, experienced users memorize the user interface layout and are able to operate applications at a reduced display size for the user

interface—e.g. smaller buttons without a caption—in order to save display size for important content. With zooming at the application window level users can freely control the display size of the entire application including the user interface such as menu bars, buttons, checkboxes, etc.

Application Content Level Resizing the content of an application window. There are numerous applications that support some kind of zooming. Unfortunately, this functionality is not standardized and each application provides a different look&feel for it. Zoomable user interfaces with application content level zooming provide a novel technique to support zooming of the application's content by the underlying windowing system in an application independent way. See Figure 5.4 for an exemplary illustration.

Context-Aware Application Switching

As stated earlier, the number of different applications concurrently used by users that are working on computers will further increase with high probability in future. Each application is designed to handle specific tasks where the output can be used as content within other applications. A major task for window managers already is and will be to efficiently support working with multiple applications. Most research in this field have been done in the infrastructure technology, reaching from Object Linking and Embedding (OLE) to Compound Documents, where content of different applications are combined within a single document. In contrast, working and interacting with these technologies has not been a focus of research and can be greatly enhanced via a zooming user interface. The application content level zooming can be extended to support context-aware application switching in a way that a user smoothly switches between applications, dependent on the level-of-detail the content is viewed.

Today's compound documents or OLE technologies already allow editing particular content objects via dedicated applications within the surrounding document. However, this is often not practical as the working space and user interface of the embedded application is too small and users, therefore, tend to decouple the application from the surrounding document to enlarge the working area. For a zooming user interface, which supports three-level zooming, a user can simply zoom the content or the entire desktop to focus on an embedded object in order to have enough working space for operating the application. The windowing system can continuously evaluate the current context by observing the utilized screen space of the embedded objects and automatically switch from the content view to an embedded application. This way, users can work with different applications within a single document and transparently switch between them in a hierarchical tree of embedded content objects. Figure 5.5 shows an example document in a zoomed out view (top left), a zoomed in view (top right), and the corresponding object hierarchy (bottom). The compound document embeds a bitmap graphic and an illustration, whereas the illustration itself contains a bitmap graphic. The object hierarchy also shows the corresponding applications that are utilized to manipulate each of them.

In summary, the presented technique makes use of context information—i.e. the application's utilized screen space—to support users in handling a great variety of advanced applications concurrently on a single desktop. The approach is naturally integrated into a zoomable user interface

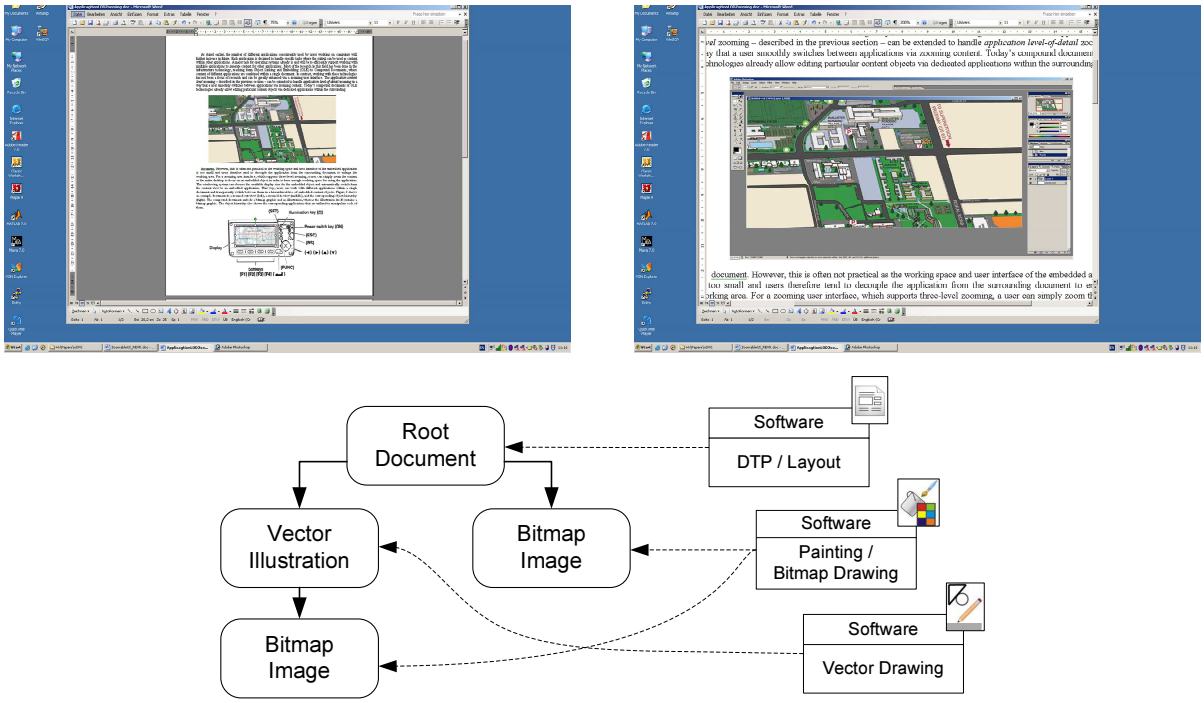


Figure 5.5: Overview of a compound document (top left), zoomed-in view for editing the embedded bitmap (top right), application/object hierarchy of a compound document (bottom).

that further allows to freely adjust the desired detail of applications, applications' content, and the entire work space.

5.2 Display-dependent Image Adaptation

In order to present computer generated images a corresponding output device is required. Numerous technologies exist that can be used: printers, cathode-ray tube monitors, liquid-crystal displays, video beamers, etc. Thereby most devices achieve optimal output quality only if the input image matches the specification of the device. Image resolution conversion is the most often required adaptation, but also color reduction for low color-depth displays is common (e.g. on mobile devices).

Independent of the applied techniques to generate images, the required storage size and—if transmitted via network—required bandwidth is also a limiting factor for today's images or videos. In fact, currently available video content, e.g. via internet streaming services, seldom matches the resolution of the utilized display device. Low-resolution content often has to be displayed on high-resolution displays. In particular data provided by webcams, camera phones, low-bandwidth video streams, and even television broadcasts require a upscale operation to match the resolution of, e.g., high definition displays (typical resolution of 1920x1080 pixels).

The same is true for remote-rendering scenarios that often require an adaptation of the provided image sequence to match the hardware resolution of the target device. In a single user

scenario images can already be generated in the corresponding resolution, but for multi-user systems images have to be used for many targets with different resolution in order to limit the computational load of the image server. Therefore, efficient high-quality scaling algorithms are required to fulfill this task. Furthermore, sending lower-resolution images to a client device can dramatically reduce the required bandwidth; however, this requires an implementation of resolution up-conversion algorithms on the client devices itself.

Besides spatial resolution also color-depth differs dependent on the utilized device. Even novel upcoming display technology may be limited in color-depth, e.g. first upcoming ePaper devices[2] can only display four different gray scales.

This shows that the demand for real-time, high-quality image-adaptation techniques is manifold and applicable to various applications.

5.2.1 Hardware-Assisted Scaling on Mobile Devices

Resolution up-conversion can be classified as a simple task, but it requires a huge amount of processing power due to the amount of data that has to be processed in a short period of time. If a video with 24 fps has to be scaled to a resolution of 480x640 pixels then about 22 MB of data has to be generated per second. As memory bandwidth and CPU processing power is very limited on mobile devices and probably already stressed, e.g., from decoding the video, embedded GPUs are preferably utilized to perform scaling tasks.

GPUs are designed for graphic processing and are, therefore, more efficient for these tasks compared to common CPUs. However, simple bilinear interpolation, as supported by all GPUs, achieves only low-quality results—especially for scaling factors > 1.5—in contrast to, e.g., CPU-based bicubic interpolation. In order to support high-quality zooming on mobile devices and still utilize the GPU, higher-order filtering schemes are examined and ported to a GPU-based implementation.

High-Quality Zooming using GPUs

An observation, already mentioned in Section 3.1.1 and presented in [54], is that most current GPUs support bilinear filtering without any performance penalty compared to a simple lookup (nearest neighbor). This fact is exploited by M. Strengert et al. in [191] to implement efficient higher-order interpolation based on multiple bilinear filter operations. This approach perfectly matches the still limited GPUs of mobile devices, since only bilinear lookup functionality is required.

The technique proposed by Strengert et al. implements a biquadratic B-spline filtering according to the Doo-Sabin subdivision schema [34]. The intended scaling factor s is divided into an up-scaling operation by a factor $u = 2^l$, where l is the smallest integer fulfilling $s <= 2^l$ and down-scaling factor $d = s/2^l$. First, the up-scaling is performed and afterwards the result is down-scaled by d using common bilinear filtering, resulting in a overall scaling by a factor of s. A single up-scaling operation enlarges the 2D source images in width and height by a factor

[2] Amazon Kindle 1: http://www.amazon.com/kindle

of two; by recursively using the result as input, scale factors of 2^x are achieved where x is the number of up-scale operations that are required.

Thereby, each up-scaling step weights the source image's pixels as shown in Figure 5.6b to calculate a higher resolution version of the source image. For comparison, Figure 5.6a shows the setup for a common bilinear interpolation, note that the biquadratic B-spline configuration (Figure 5.6b) is only an approximation of the source image.

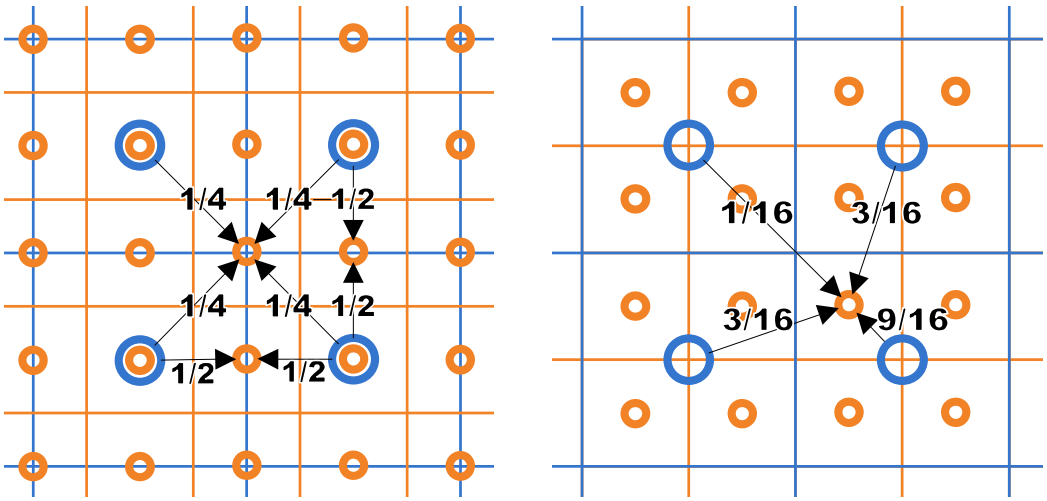


Figure 5.6: Image zooming by bilinear interpolation (a) and biquadratic B-spline approximation (b). Circles indicate pixel centers. The source data is drawn in blue; destination pixels are shown in orange.

Results of an Implementation on Mobile Devices

For performance evaluation on mobile devices of the aforementioned technique, an implementation based on OpenGL ES 1.0 was developed. The utilized device is an O_2 XDA Flame PDA embedding a Nvidia GoForce 5500 GPU running the Windows Mobile 5.0 operating system. Internally, the implementation executes multiple render-to-texture operations with texture coordinates according to the biquadratic B-spline filtering scheme. The number of recursive render-to-texture operations is dependent on the input and output dimensions: A 32x32 resolution input image requires three steps to generate a 256x256 resolution output (4 steps for 512x512). In contrast, the bilinear up-scaling is calculated in a single step, independent of input or output resolutions. Table 5.1 shows the performance for image zooming compared with a simple bilinear scaling. Note that the final output is clipped to a 640x480 framebuffer, so that only 512x480 (640x480) pixels are rendered for a 512x512 (1024x1024) output resolution; however, intermediate zoom steps of the biquadratic B-spline approach are performed using an off-screen buffer (P-Buffer) and are therefore not affected.

Table 5.1 also reflects the performance impact of using a large P-Buffer, e.g. 512x512, for off-screen rendering; however, no performance impact for the bilinear method can be observed.

Table 5.1: Performance in [fps] for an up-scaling operation on a Nvidia GoForce 5500 GPU at various setups. Invalid configurations are marked as not applicable *n/a*, unexpected slowdowns are written in bold.

P-Buffer Size of 256x256 pixels:

Source	bilinear filtering			biquadratic B-spline filtering		
Image Size	256^2	512^2	1024^2	256^2	512^2	1024^2
16x16	98.2	86.7	83.4	83.5	31.3	*n/a*
32x32	98.2	86.7	83.3	**65.5**	74.1	*n/a*
64x64	97.8	86.4	83.4	95.9	**32.5**	*n/a*
128x128	96.3	85.2	82.9	96.3	81.2	*n/a*
256x256	*n/a*	**52.7**	81.2	*n/a*	**52.7**	*n/a*

P-Buffer Size of 512x512 pixels:

Source	bilinear filtering			biquadratic B-spline filtering		
Image Size	256^2	512^2	1024^2	256^2	512^2	1024^2
16x16	98.3	86.7	83.4	60.2	28.3	20.2
32x32	98.1	86.8	83.3	62.4	28.7	20.4
64x64	97.9	86.4	83.4	65.5	29.0	20.5
128x128	96.3	85.6	82.9	96.2	29.3	20.7
256x256	*n/a*	**54.8**	81.2	*n/a*	**54.8**	21.5

This shows that allocations of large P-Buffers do not affect other parts of the GPU, as could have been possible due to memory requirement. In contrast, if larger P-Buffers are used during the rendering—as for the biquadratic B-spline case—huge performance degradations can be observed. Furthermore, if the biquadratic B-spline case executes only one zoom iteration, the implementation can directly write the result of the zooming operation into the framebuffer without utilizing an off-screen buffer. In these cases, both techniques achieve the same performance at comparable quality (e.g. 128x128 input resolution up-scaled to 256x256 output resolution). The same resulting image quality as presented in [191] is achieved; some examples are shown in Figures 5.10, 5.11.

Even though the measurements show some configurations that are unexpected slow, like for a small P-Buffer a zooming from 64x64 up to 512x512, the biquadratic B-spline filtering technique is still adequate for high-quality interactive zooming on mobile devices, even for full-screen resolutions. At the same time there is no load distributed to the CPU, which is now free to be used for image decoding or any other task.

5.2.2 Training-based Approaches for Resolution Up-Conversion

An entirely different approach for image and video stream up-scaling is the class of learning or training-based methods. The basic approach is to examine the content of a low-resolution image to generate a high-resolution image out of it. During the up-scaling, previously trained knowledge is applied resulting in a content-aware resolution up-conversion technique.

A major advantage is that the method is locally adapted due to the source-image content and is, therefore, able to handle different image regions, e.g. sharp details of some physical structure in an image where smooth clouds are visible in the sky. This advantage makes it especially appropriate for consumer television systems where arbitrary content is displayed. However, such devices embed a hardware-based implementation, dedicated for resolution up-conversion to cope with the high demands of processing power of such algorithms (as presented in [120]). Devices that lack such a hardware component, e.g. laptop, desktop pc, game console, etc., have therefore often a reduced image quality when low-resolution content is displayed on a connected high-resolution display.

The results of a diploma thesis [143] on the analysis of existing training-based up-conversion techniques and new prototypical GPU-based implementations directly address this limitation. The diploma thesis was performed by J. Nausedat [143] and supervised in cooperation with M. Strengert and the European Technology Center (EuTEC) of Sony Germany. The work concentrates on the implementation of a DRC-like approach with the goal to reproduce results of the Digital Reality Creation technology by Sony. Note that the original DRC algorithm might perform various steps differently and, thus, might achieve an even better image quality; however, the prototype implementation is according the basic approach as documented in [120].

The algorithm examines a 3x3 pixel area of the source image for each input pixel, i.e. the 8-neighborhood of each source pixel. Based on these 9 input pixel—defining the local content—a classifier calculates a class for the source pixel, as illustrated in Figure 5.7. The classification process is further detailed in the following subsection. As shown in the figure, the determined class is used to access the table of trained filter weights. The table stores for each possible class a set of 9 filter weights per subpixels that get interpolated, i.e. four times nine filter weights per class in total. The training process to generate the filter weights is a preprocessing step and explained in a subsequent paragraph.

The selected filter weights are used by the interpolation to weight the source pixel data and calculate four subpixels for the currently processed input pixel in parallel. In Figure 5.7 the interpolation is only shown for subpixel A. All other subpixels are calculated analogously but with different weights. The interpolation of the luminance of subpixel A (L_A) is therefore evaluated according to Equation 5.1.

$$L_A = \sum_{m=0}^{2} \sum_{n=0}^{2} w_A(m,n) L(m,n) \tag{5.1}$$

Selected filter weights w_A are organized in a 3x3 pattern and addressed with the indices m, n; the 3x3 low-resolution input area is denoted with L. For color source images, the interpolation is executed for each color component separately, but using the same filter weights to avoid color

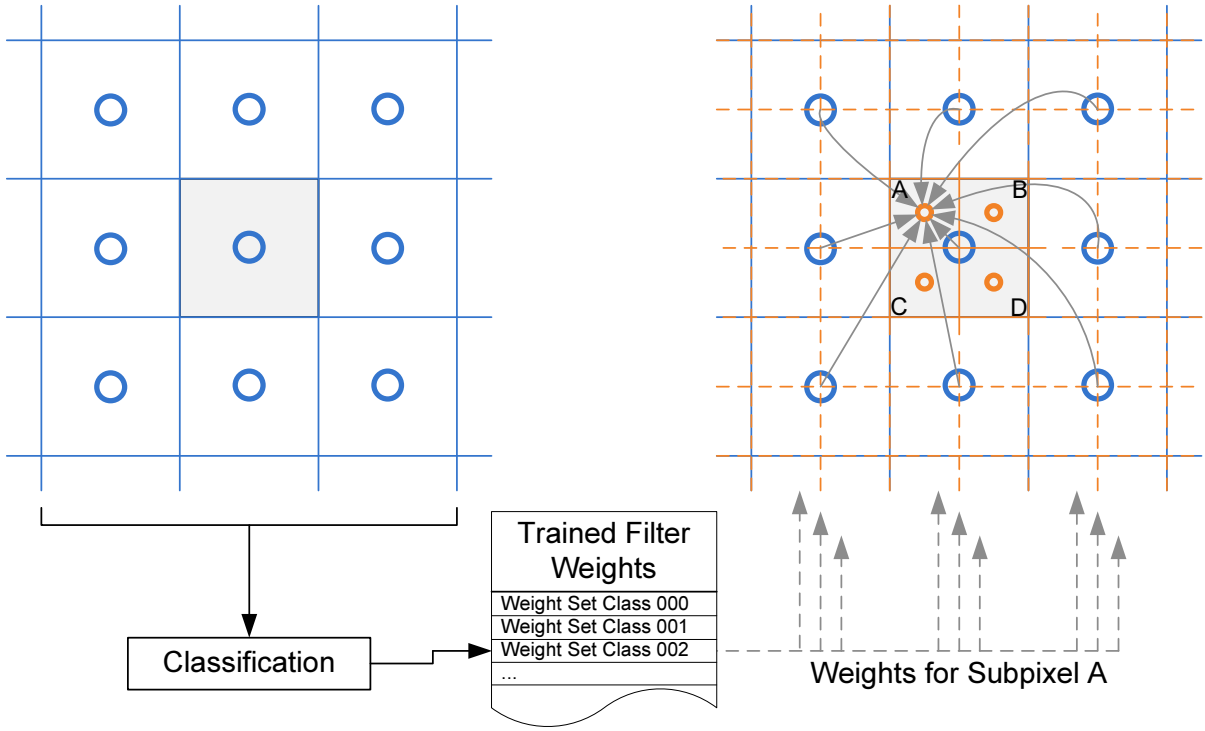


Figure 5.7: Overview of the DRC-like approach. The light gray pixel is processed and subdivided into four subpixels (A, B, C, and D) concurrently. The classification is based on a 3x3 pixel area around the currently processed input pixel and selects a set of 4x9 filter weights to interpolate the four high-resolution subpixels. Note that the illustration shows the final interpolation only for subpixel A.

shifts of pixels.

The obtained quality of the approach is strongly dependent on the classification process of the content and the pre-trained filter weights.

Local Content Classification

Classification of 3x3 input pixels corresponds to finding a mapping of 3x3 source pixels to k pixel classes. A naïve implementation would map each configuration to a unique class. However, this would result in an impracticable amount of classes, since already for scalar 8 bit input values $256^9 = k$ different configurations are possible. Furthermore, for each class, 28 filter weights have to stored, resulting in an unrulable amount of memory. Therefore, the most important task is to reduce the number of classes dramatically, thereby sufficiently different content still has to be mapped to distinct classes.

For the prototypical implementation, realized within the diploma thesis [143], the classification is based on Adaptive Dynamic Range Coding (ADRC) [119], as suggested by T. Kondo in [120]. Using ADRC each input pixel is compressed to 1 bit resulting in $2^9 = k$ classes. The remaining 1 bit information expresses if the corresponding pixel is above the average luminance

of the 3x3 area or below. In addition, inverse configurations of the reduced content description can be mapped to the same class, so that the total number of classes used in the prototype is $2^8 = 256$.

Training of Filter Weights

For every possible class, addressed by the utilized classifier, filter weights have to be calculated (via training) and stored in a look-up table. Training is discussed for an up-scaling factor of two as required to scale standard-definition television (SDTV ∼960x540 pixels) content up to be displayed on a full high definition television (HDTV 1980x1080 pixels). Using the same basic principal, also other zoom factors are possible—e.g. if each source pixel is subdivided into 3x3 subpixels; however, a larger look-up table for the weights and a different training is then required.

The training process makes use of full-resolution reference material which is down-scaled by a factor of two to acquire low-resolution (LR) input content and its corresponding optimal up-scaling result in high resolution (HR). On the LR content the aforementioned ADRC classifier is executed for each pixel within each frame of the training material to determine the input pixel's class as depicted in Figure 5.8. During this process, an intermediate table for training sets is used to store for each input pixel: the class of the pixel, the pixel values of the 3x3 area of the source content, and the corresponding 2x2 high-resolution pixels copied from the HR content.

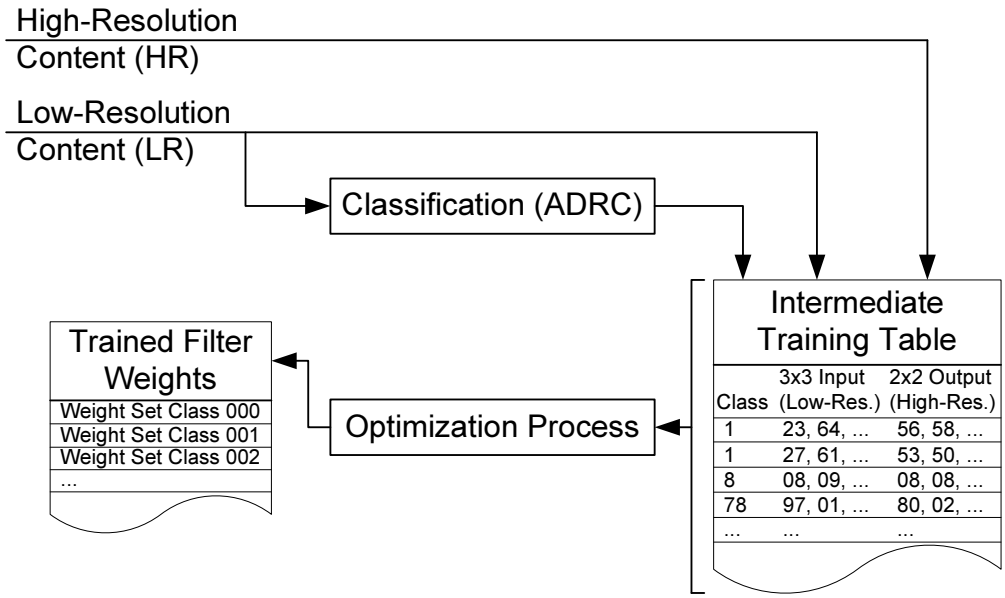


Figure 5.8: Training of the filter weights. This pre-processing step is divided into two parts: First an intermediate table of corresponding LR and HR samples is build; afterwards, this table is used as input for an optimization approach to calculate the final weights.

After processing all training material, the intermediate training table holds multiple sets of input and corresponding output pixels for each class. For the final calculation of a single set of filter weights per subpixel within each class, an optimization approach is used to optimize

the weights based on all acquired LR-HR combinations and stored in the intermediate training table. Therefore, a system of equations is setup for each subpixel within each class. Equation 5.2 illustrates the system for a single subpixel where $l_{1..n,1..9}$ corresponds to the n recorded low-resolution 3x3 areas and $h_{1..n}$ to the associated reference high-resolution results. The resulting filter weights are $w_{1..9}$. Note that the 8-neighborhood and the filter weights are defined as a vector.

$$\underbrace{\begin{pmatrix} l_{1,1} & \dots & l_{1,9} \\ \vdots & \ddots & \vdots \\ l_{n,1} & \dots & l_{n,9} \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} w_1 \\ \vdots \\ w_9 \end{pmatrix}}_{\vec{w}} = \underbrace{\begin{pmatrix} h_1 \\ \vdots \\ h_n \end{pmatrix}}_{\vec{h}} \tag{5.2}$$

In general, this system of equations is over-determined and, therefore, an optimization approach is used to find the *best* filter weights. By introducing an error vector $\vec{e}$ a multidimensional minimization algorithm can be used to minimize the total error by finding optimal filter weights $\vec{w}$, as shown in 5.3.

$$\begin{aligned} \vec{e} &= L\vec{w} - \vec{h} \\ \min_{\vec{w}} &\sum_{i=0}^{n} e_i^2 \end{aligned} \tag{5.3}$$

For the implementation a simple Simplex-Downhill [144] method to optimize the multidimensional problem was used, as the performance of the pre-processing step is irrelevant. After executing the optimization for each subpixel within all classes using sufficient LR-HR training material—in order to acquire enough LR-HR references for all possible classes—the table of trained filter weights can be used by the DRC-like algorithm.

Implementation on Graphics Hardware and Results

As motivated, the goal of the diploma thesis [143] was to implement a fast, high-quality GPU-accelerated up-scaling, based on the DRC algorithm. As the training of filter weights is a pre-processing step, this part is implemented using the CPU only.

In contrast, the actual up-scaling is implemented on graphics processing hardware: The method computes the final interpolation in two phases; both are implemented as fragment programs. Due to GPU restrictions, the first pass makes use of multiple render targets to simultaneously output four high-resolution pixels per low-resolution input pixel. Fragment processing starts by reading the entire 3x3 input area via 9 texture look-ups, followed by the average computation and classification. For evaluation of the finally resulting high-resolution subpixels, texture look-ups—dependent on the subpixel (A, B, C, or D) and class—into the table of trained filter weights are performed to retrieve the appropriate weights, dependent on the source content. These weights are then used to calculate the weighted sum, as expressed in Equation 5.1, per subpixel. In a second pass, a minimal fragment-shader program combines the four independent

render targets of the previous pass to a higher-resolution output target with the size of two times the input resolution in both dimensions.

For the examined configuration, when scaling SD content for display on a HD device, the prototypical GPU implementation performed at approximately 100 fps using an ATI Radeon X1900 graphics card, without noticeable CPU load. However, the qualitative result of this first prototypical implementation is visually comparable to bicubic interpolation results, but it was also recognized that the results are strongly dependent on the trained filter weights. Therefore, it is expected that an improved training with more and better training material will result in a better quality at the same performance.

Still, the result motivates future work since it was shown that the basic approach could efficiently be mapped to graphics processing units, even leaving some GPU resources free for further optimizations. An interesting enhancement is presented by H. Hu and G. de Haan [98] to achieve coding-artifact reduction and sharpening of the up-scaled results. The sharpening is achieved by simply training with blurred down-sampled material. Similarly, training source images were modified with coding artifacts, so that the filters coefficients are trained to remove these. This method shows the advantage of using training-based approaches as the same basic algorithm is now able to remove coding artifacts, sharpen images, and perform the up-scaling with only marginal changes of the original up-scaling implementation.

5.2.3 Content-Aware Image Zooming on GPUs

Similar to training-based approaches, as presented in Section 5.2.2, specific content features can be extracted and used to adapt algorithms according to the data content that is processed. This way, a higher quality of up-scaled images can be achieved.

A correct reconstruction of image data could be computed using an ideal sinc filter, but this requires a sampling of the image data above the Nyquist frequency, which is in practice not possible. Reconstruction using a sinc filter on undersampled data result in ringing artifacts (Gibbs phenomenon); therefore, an additional low-pass filter is often applied to remove the ringing. Unfortunately, this low-pass filtering introduces noticeable blurring of sharp edges in zoomed images. Human observers often have further knowledge of the displayed content and can therefore much better estimate the correct sharpness of an edge. Zoomed illustrations of real-world sharp edges are therefore perceived as wrongly blurred and classified as artifacts. Even high-quality approaches like, e.g., the biquadratic B-spline zooming suffer from this problem.

The well-known problem of high-quality image up-sampling has driven countless research efforts. Model-based approaches are proposed in [5] to remove the excessive blurring by reconstructing the original physical image signal based on a proposed observation model. X. Li and M.T. Orchard present an approach that controls the interpolation utilizing the geometric duality between the low-resolution covariance and the high-resolution covariance [126]. A technique to locally select the applied interpolation scheme based on edge detection, in order to reduce jagging artifacts in up-scaled edges, is proposed by Q. Wang and R. Ward in [200]. They further propose to apply a sharpening function in edge areas.

A further technique to reduce the blurring of actually sharp edges designed for real-time applications using a GPU-based implementation is proposed in collaboration with M. Kraus and

M. Strengert in [122]. The basic idea is to improve the image quality of zoomed images by controlling the sharpening of edges, based on the image content. In particular, edges in the image are considered to adjust the up-scaling process. A model for an ideal edge, as applied in the work of G. Kindlmann and J. Durkin in [114], is used to define an edge and characterize its transition region.

Definition of an Ideal Edge for Edge Preservation

A physical signal s of an ideal edge is assumed to feature an arbitrarily sharp change from y_{min} to y_{max} at position x_e, as illustrated in Figure 5.9a. The mathematical formulation is a scaled and shifted step function Θ:

$$s(x) = y_{min} + (y_{max} - y_{min})\Theta(x - x_e). \tag{5.4}$$

When measuring such a physical signal, blurring occurs which can be modeled as a convolution of the physical signal s with a normal distribution with standard deviation σ (see Figure 5.9b). The resulting sampled signal f is plotted in Figure 5.9c and defined as:

$$f(x) = (y_{min} + (y_{max} - y_{min})\Theta(x - x_e)) \otimes \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\left(\frac{x^2}{2\sigma^2}\right)}. \tag{5.5}$$

By performing the convolution the following result is obtained:

$$f(x) = y_{min} \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\left(\frac{\tau^2}{2\sigma^2}\right)} \,\mathrm{d}\tau + (y_{max} - y_{min}) \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\left(\frac{\tau^2}{2\sigma^2}\right)} \Theta((x-\tau) - x_e) \,\mathrm{d}\tau. \tag{5.6}$$

As the step function within the second integral of Equation (5.6) is zero for all values of $\tau > (x - x_e)$ and one otherwise, the convolution can be solved by adjusting the integral bounds as follows:

$$f(x) = y_{min} \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\left(\frac{\tau^2}{2\sigma^2}\right)} \,\mathrm{d}\tau + (y_{max} - y_{min}) \int_{-\infty}^{x-x_e} \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\left(\frac{\tau^2}{2\sigma^2}\right)} \,\mathrm{d}\tau. \tag{5.7}$$

Further solving the function f results in a parameterized error function (*erf*) which finally describes the sampled signal of an ideal edge (see Figure 5.9c):

$$f(x) = \frac{y_{max} + y_{min}}{2} + \frac{y_{max} - y_{min}}{2} \mathit{erf}\left(\frac{x - x_e}{\sigma\sqrt{2}}\right). \tag{5.8}$$

Using the following definition for the error function (*erf*):

$$\mathit{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \mathrm{e}^{-\tau^2} \,\mathrm{d}\tau. \tag{5.9}$$

The exact position of the ideal edge x_e can be calculated based on the first or second derivatives of the function f, by finding the maximum extreme of f' or the zero crossing of f'' as illustrated in Figure 5.9d. Well-known edge detectors by J. Canny [33] or D. Marr and E. Hildreth [134] also make use of these characteristics.

$$\begin{aligned} f'(x) &= \frac{y_{max} - y_{min}}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\left(\frac{x-x_e}{\sigma\sqrt{2}}\right)^2}. \\ f''(x) &= -\frac{(x - x_e)(y_{max} - y_{min})}{\sigma^3\sqrt{2\pi}} \mathrm{e}^{-\left(\frac{x-x_e}{\sigma\sqrt{2}}\right)^2}. \end{aligned} \tag{5.10}$$

Both functions are combined to form a scale-invariant characterization function d, which can be used to describe the transition region of magnified edges. Equation (5.11) shows the characterization function and its property that a zero crossing of d corresponds to the edge position x_e.

$$d(x) \stackrel{\text{def}}{=} \frac{-\sigma^2 f''(x)}{f'(x)} = x - x_e. \tag{5.11}$$

Note that the function d is defined independently of y_{max}, y_{min}, and x_e and is therefore especially appropriate to control the edge-sharpening process in a magnification method.

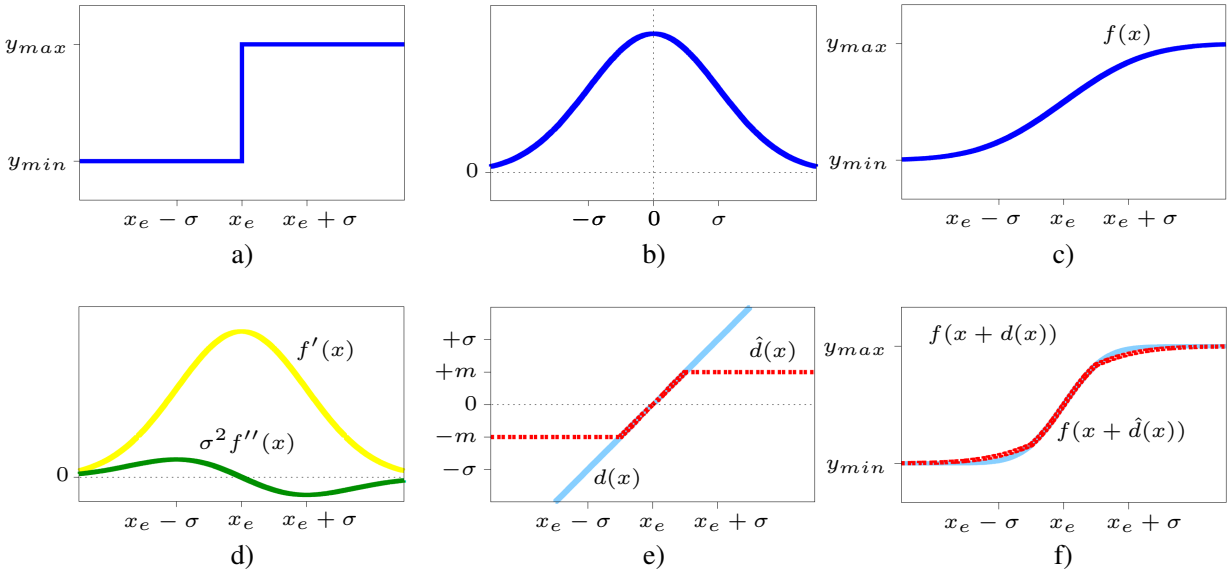


Figure 5.9: Detecting and sharpening of an ideal edge signal shown in a). The filter applied when measuring the signal is modeled with a normal distribution as depicted in b); c) shows the resulting sampled ideal edge signal. First and second derivative (scaled by σ^2) of this sampled signal is plotted in d). The offset $d(x)$ and clipped offset $\hat{d}(x)$ (dotted line) applied when resampling is seen in e); the resampled and sharpened signal $f(x + d(x)$ and $f(x + \hat{d}(x))$ (dotted line) is shown in f).

Edge-Controlled Magnification

Using the model of an ideal edge and its sampled signal—defined via the function f—the process of magnification can be further examined. If the sampling resolution is increased by a factor of two the resulting signal should be twice as sharp as illustrated in Figure 5.9f. This is achieved if the convolution in (5.5) is performed with a normal distribution with half the standard distribution $\sigma/2$. Equations (5.12) show the function h that models this higher-resolution signal, which can also be expressed with the previous function f.

$$\begin{aligned} h(x) &= \frac{y_{max}+y_{min}}{2} + \frac{y_{max}-y_{min}}{2} erf\left(\frac{2(x-x_e)}{\sigma\sqrt{2}}\right). \\ h(x) &= f(2x-x_e) = f(x+(x-x_e)) = f(x+d(x)). \end{aligned} \tag{5.12}$$

Therefore, if the signal is scaled by a factor of two a sharpening of an ideal edge can be achieved by simply sampling the signal with a positional offset given by d. However, the presented analysis is only valid for an ideal edge. For arbitrary signals the numerical computation of d gets unstable and can result in huge offsets that cause strong artifacts in the magnified signal. In order to prevent such cases, the result of d is clipped to a boundary of $[-m,+m]$:

$$\hat{d}(x) = max(-m, min(+m, d(x))). \tag{5.13}$$

The influence of the clipping is shown as a dotted line in Figure 5.9e. Limiting the offset stabilizes the computation even for very large values of d, e.g. in cases where $f'(x) \to 0$.

Although the computed offset $\hat{d}$ for the sampling positions can be applied to arbitrary image-zooming methods, only the biquadratic B-spline approach by M. Strengert et al. [191]—as detailed in Section 5.2.1—was examined as basic zooming technique since it provides already good results and allows an efficient implementation using graphics hardware. Depending on the sampling scheme of the utilized zooming approach, the boundary clamping of $\hat{d}$ has to be adjusted with the parameter m (for a biquadratic B-spline sampling scheme m should be chosen between 0.25 and 0.5).

To compute $\hat{d}$ in a two-dimensional domain, the first and second derivatives have to be computed across edges; thus in the direction of the gradient $\nabla f(x)$. For the prototypical implementation, $f'(x)$ is computed using central differences and $f''(x)$ is approximated by the Laplacian $\Delta f(x)$ in the direction of the gradient, as suggested in [114]. An efficient GPU-based implementation to calculate $\hat{d}$ by utilizing multiple bilinear interpolation operations is presented in [122].

Results of GPU-based Up-Sampling Implementation

In contrast to the implementation of biquadratic B-spline zooming, an implementation of the proposed content-controlled zooming technique requires GPU support for dependent texture lookups and fragment programs (see Section 3.1). Although these requirements are low and all of today's desktop GPUs are supported, an implementation on mobile devices is not possible using the standard OpenGL ES 1.x API and requires OpenGL ES 2.0. Due to the lack of OpenGL ES 2.0

hard-/software the evaluation was performed using an implementation for desktop GPUs using OpenGL 2.0.

Qualitative results for grayscale images of the GPU-based implementation are shown in Figure 5.10 (middle column). The results show the quality of the algorithm for zooming factors of 2, 4, and 8 (top to bottom). For reference, results of different methods are also presented (left column: biquadratic B-spline, right column: bicubic). The image source is a detail of the ISO 12233 Resolution Test Chart.

Figure 5.10 illustrates how the proposed approach handles oblique edges in contrast to other approaches. The biquadratic B-spline filtering introduces a noticeable blur; the bicubic approach shows better results with less blurring but higher computational costs. The edge-controlled enhancement—shown in the middle column—shows sharp features with only marginal blurring. However, at edges with an L-shaped profile, small ripples are introduced.

In order to apply the edge-controlled filtering for color images, the same value of $\hat{d}$ has to be used for all three color components to avoid color shifts that might occur if different resampling positions are calculated for the red, green, and blue channel. The prototype therefore converts color images to luminance images for the calculation of $\hat{d}$; the up-scaling is then performed for all three color components equally. The qualitative results are depicted in Figure 5.11, where the algorithm was setup with $\sigma = 0.7$ and $m = 0.25$.

Table 5.2: Performance in [ms] of the edge-controlled up-scaling technique. Viewport size for measurement was 1024x1024.

	zoom factor						
GPU Type	x2	x4	x8	x16	x32	x64	x128
Nvidia 6800GT	3.91	5.37	5.82	6.00	6.02	6.06	6.08
Nvidia 8800GTX	0.70	0.95	1.04	1.10	1.12	1.14	1.17

The performance of the OpenGL 2.0 based prototype implementation is presented for various zoom levels on different GPUs. All results shown in Table 5.2 were measured for a target resolution of 1024x1024 pixels. As verified by the measurements, the proposed technique is easily capable to up-scale high-resolution video content for real-time applications. Even on low-performance GPUs (NV 6800GT) with less available memory bandwidth the achieved performance is still adequate.

5.2.4 Coherent Halftoning for Low Color-Depth Displays

When supporting presentation of various visualization results on arbitrary display devices not only limitations of the spatial resolution of output devices have to be considered but also restrictions in the number of colors or intensity levels a display can reproduce. Especially in mobile scenarios several, even newly developed, display technologies can only present a very limited color range or only few intensity values. Such limitations are serious for data visualization, since

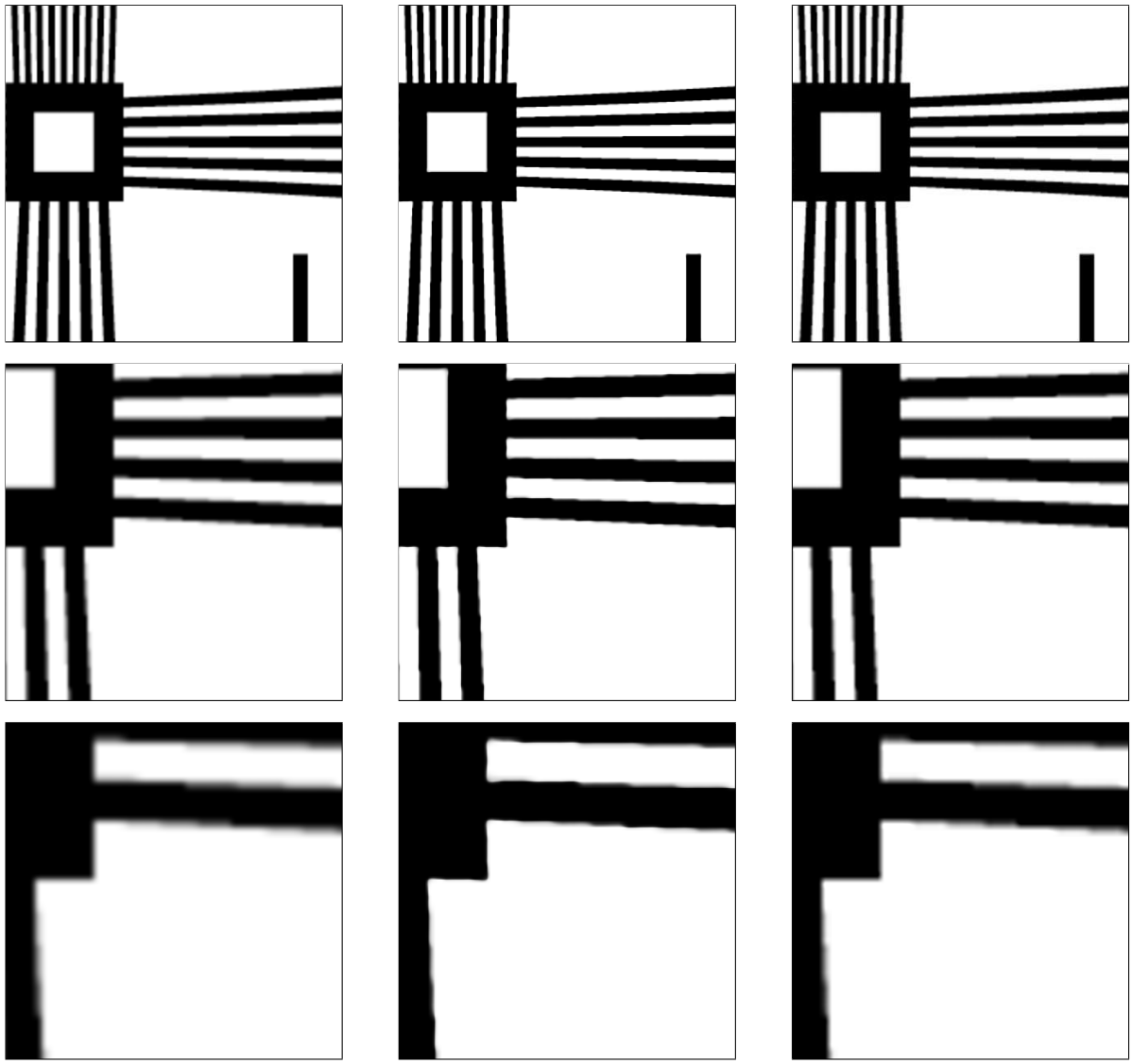


Figure 5.10: Results of up-scaling a grayscale image by a factor of 2, 4, and 8 (top to bottom). The utilized algorithms are the biquadratic B-spline method (left column), the proposed edge-controlled zooming (middle column, $\sigma = 0.75$, $m = 0.25$), and bicubic interpolation (right column).

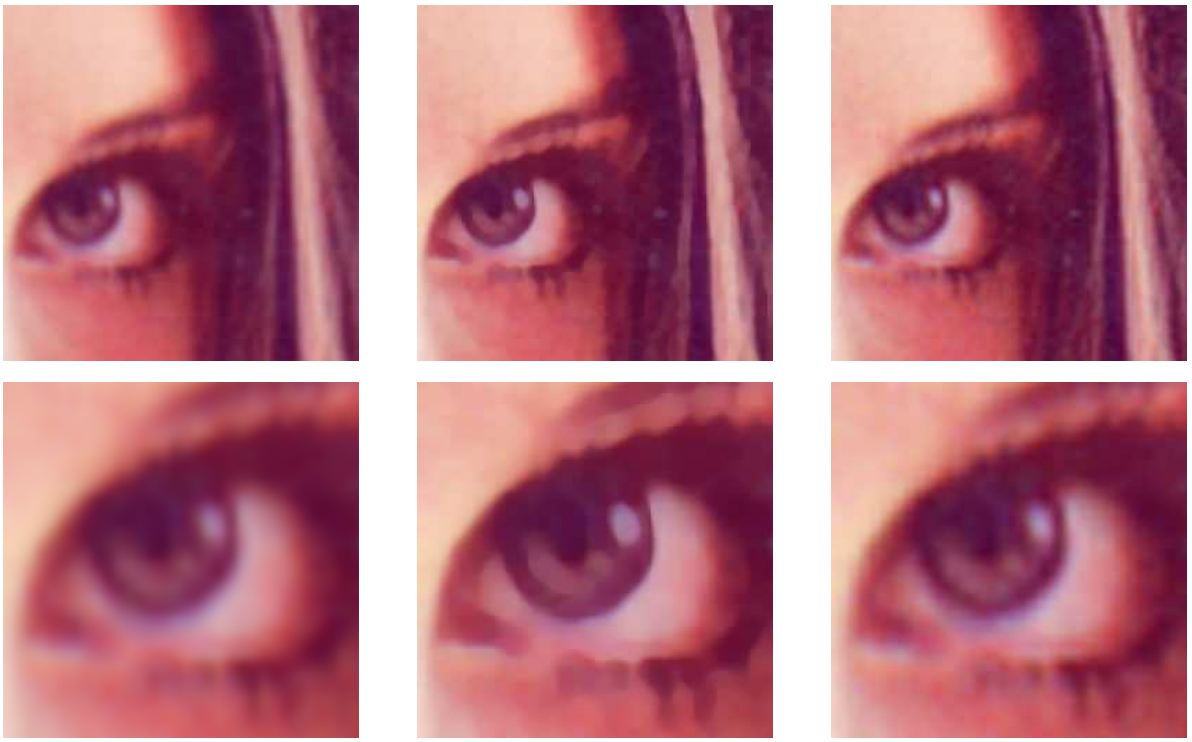


Figure 5.11: Up-scaling results of color images. The top row illustrates scaling by a factor of 4, bottom row shows results for a factor of 8. The utilized algorithms are the biquadratic B-spline method (left column), the proposed edge-controlled zooming (middle column, $\sigma = 0.75$, $m = 0.25$), and bicubic interpolation (right column).

often changes in color/intensity are used to convey attributes of the underlying data. Table 5.3 lists some examples of display devices that show aforementioned constraints.

Although pico projectors support full-color output these devices have still limitations when applied for visualization. Analogous to common video-image projectors a good quality of the projected image—in terms of color, brightness, and contrast—can only be achieved if the environmental lighting is relatively dim. Otherwise, only colors with high intensity are visible severely limiting the total amount of *usable* colors. Therefore, even devices that theoretically support full-color output may benefit from low color-depth visualization support in specific situations.

A general approach to compensate for this lack is to trade off resolution for an increased color/intensity range. This process is called *halftoning* and is extensively used in print media such as newspapers or magazines but also by printers.

However, for animated, interactive applications common image-space halftoning produces so-called shower-door artifacts [137], which closely describes the perceived image, as the halftoning pattern is fixed in image space whereas features of the scene freely move behind it. For interactive applications, such a behavior distracts users, and therefore, has to be removed. A coherent image-space halftoning approach to overcome this limitation is presented in coopera-

Table 5.3: Technical data of mobile display products. Properties that limit the usage for visualization are shown in bold.

	Resolution	Color	Intensity Range
Retina Display[3]	800x600	**monochrome (red)**	**5 bit**
Pico Projector[4]	480x320	full color	(24 bit ?)
Ink e-Paper[5]	600x800	**monochrome (black)**	**4/2 bit**

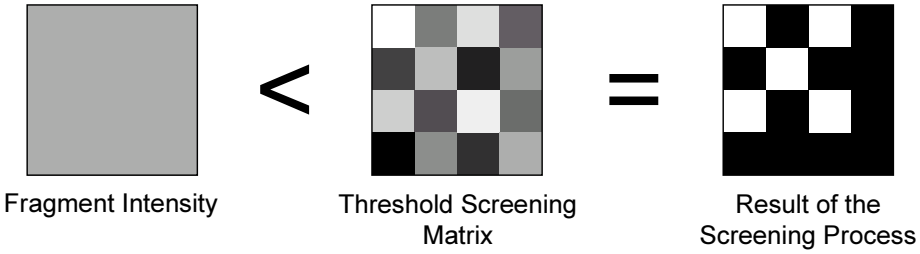


Figure 5.12: Threshold screening technique.

tion with D. Weiskopf [61]. The technique transports the halftoning patterns in a frame-to-frame coherent manner according to the velocity of each fragment on the image plane. The algorithm is mapped to programmable graphics hardware to achieve interactive frame rates and can therefore be applied in interactive applications.

Most previous approaches make use of object-space information. E. Praun et al. propose a interactive hatching technique, implemented using graphics hardware [162]. Based on this object-space approach, A. Baer et al. propose a coherent stippling approach to alleviate artifacts when parts of a 3D scene are zoomed (scaling or changes in distance) [9]. A hybrid approach is recently proposed by S. Breslav et al. where a screen-space (hatching) pattern is moved according to the underlying scene [25].

Results of the recently proposed approaches are impressive but again object-space information is used to transport the screen-space pattern and therefore the techniques cannot easily be implemented as a simple postprocessing step. In contrast, the technique proposed here limits itself to image-space attributes to allow an implementation as postprocessing filter within a generic framework.

Halftoning with Transported Screening Masks

Halftoning can be achieved with a screening technique where gray-scale or color images are combined with an underlying screening mask to obtain the final rendering. Figure 5.12 shows a

[3]Microvision Nomad: http://www.microvision.com
[4]Samsung MBP200 Pico: http://www.samsungusanews.com
[5]Amazon Kindle 1/2: http://www.amazon.com/kindle

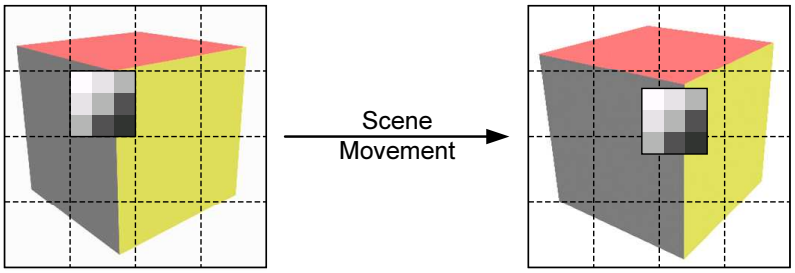


Figure 5.13: Advection of a screening matrix.

typical example for a screening technique: threshold screening. Here, the gray value of a fragment from the original picture is compared to entries in a screening matrix; the final pixel is black if the original gray value is below the corresponding entry, otherwise white. In threshold screening the screening mask, which covers the complete image space, is decomposed into smaller matrices. These screening matrices are located on a uniform grid, filling the screening mask. Thus, the position of elements within the screening mask is independent of the motion of the underlying scene; for this reason, previously known screening techniques suffer from the shower-door effect. This artifact is already distracting in desktop applications, but it is even more severe when working with head-mounted displays since users are distracted by the *fixed* grid of the screening patterns.

To overcome this fundamental drawback of image-space halftoning techniques, an approach is proposed that advects the screening mask in image space according to the velocity of each pixel in the image plane. This way, a coupling of the screening mask and the moving objects of the scene is established. Figure 5.13 shows two snapshots taken from a rotating cube to illustrate how a single screening matrix is moving with the underlying object. The image-space velocities of fragments—in the context of computer vision also referred to as optical flow [83, 93]—can either be calculated directly from the 3D scene, e.g. by using rigid-body kinematics, or from video sequences [93].

By using advected screening masks, the following issues are introduced: First, screening matrices may move to positions that are no longer aligned with the underlying grid, which is shown in the right image in Figure 5.13. A second problem is caused by velocity fields that contain regions of divergence, convergence, and discontinuity (*distortion areas*). For example, divergence and convergence appear in the case of a rotating object or camera zooming during the animation. For these areas no optimal solution can be calculated, since image-space advection of screening masks is subject to the following contradicting requirements: (1) screening matrices should have equal size and be aligned to an underlying grid and (2) screening matrices should accurately follow the motion of the objects.

To account for both cases, multiple screening masks are introduced, each with the same size as the image plane: (1) a static, non-advected mask C, which accounts for screening matrices aligned to the underlying grid; (2) a screening mask S, which is advected to follow the motion of the objects, and (3) a so-called *blend-in screening mask* B, used to couple the above two masks and make possible a smooth insertion of new screening matrices.

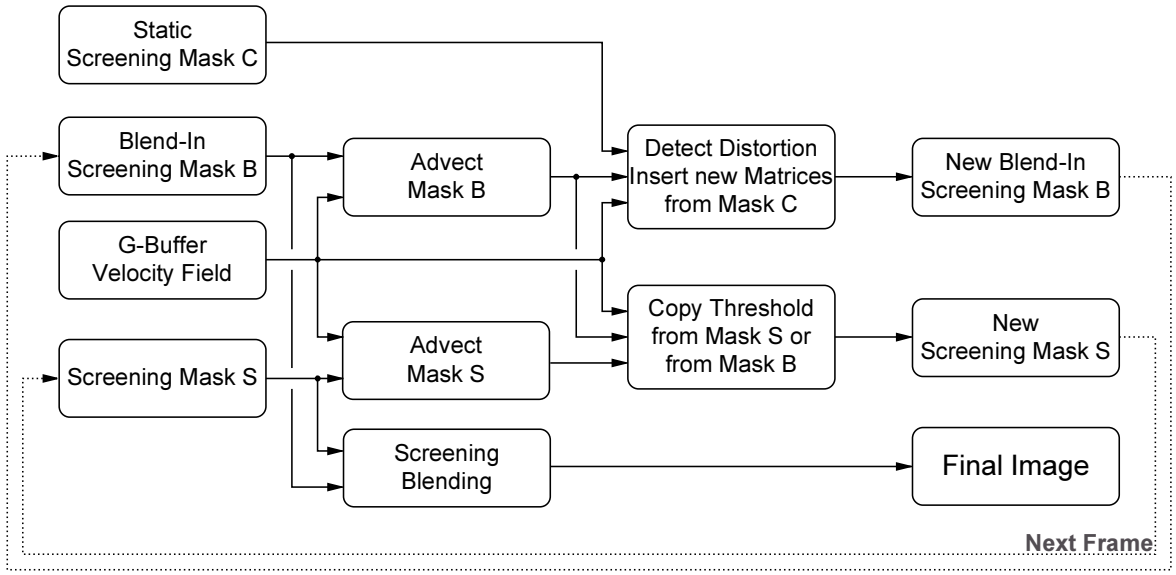


Figure 5.14: Calculation scheme for one time step of an animation.

The proposed coherent halftoning process is subdivided in three steps: Calculation of the final image, update of the blend-in screening mask B, and update of screening mask S. Figure 5.14 illustrates the flow chart for the combination of all three tasks.

To generate the final image, the results of the screening process with the mask S and the blend-in mask B have to be combined. Therefore, each fragment of the screening mask S has appended status information that consists of the total advection distance in x- and y-direction. Additionally, the blend-in screening mask B holds a timer that is incremented each frame and thus reflects the age of a fragment in the B mask. The maximum of both pieces of information is used to determine the weight for blending the screening results of the screening mask S and the blend-in mask B to generate the final image. This ensures that the fragment is completely blended when it has either been advected by the size of one screening matrix in x- or y-direction or the maximum time for the blending has elapsed. The timer restricts the duration of the blending process to a user-specified threshold, while the direction counters accelerate the blending depending on the advection velocity.

For updating the blend-in screening mask B the algorithm advects B according the velocity values in the G-Buffer attribute—as detailed in Section 3.1.3—and increments the timer for each fragment. A backward Lagrangian-Eulerian advection (LEA) scheme [104] is applied to transport elements of B by displacing them at each time step. LEA makes use of nearest-neighbor sampling in the backward advection lookup to maintain the contrast of the original input texture and to preserve the structure of the screening mask. Unfortunately, a straightforward implementation of advection with nearest-neighbor sampling would result in an animation that is partitioned into regions of constant motion, within which the integer part of the displacement is constant. This unpleasant quantization of the velocity vectors is described in more detail in [104]. Fractional coordinates are used to overcome this problem: These additional coordinates determine the subtexel position of each element of the input texture and are updated for each time step

according to the velocity field. In this way, subtexel motion is possible and the animation is no longer partitioned into regions of quantized velocities.

The primary purpose of the blend-in mask is to detect distortion regions in the velocity field and schedule these for a replacement in the blend-in mask B with new matrices from the static screening mask C. As a measure for the distortion, the difference of the velocity at the current fragment position and at the previous position is considered. Unfortunately, the velocity field can contain a wide range of velocity values and therefore simple thresholds for finite differences are not appropriate. Thus, our implementation actually computes the ratio of both velocities. If the ratio is below a given divergence threshold, the fragment is assumed to be located in a divergence region and marked for overwriting with the static screening mask C. A ratio above the convergence threshold also initiates an overwrite of the fragment. Additionally, the distance counters are examined to check whether the fragment has been advected for more than the extent of the screening matrix, which also initiates an overwrite by the value of the static screening mask C.

To ensure that only entire matrices are inserted, the distortion regions have to be exactly aligned with the layout of the static screening mask C. This is achieved by a guided dilation that marks the entire matrix if at least one containing fragment is marked. All fragments of a newly inserted screening matrix receive an initial value of 0.0 for both the timer and the distance counters. Along subsequent frames, the contribution of the blend-in screening mask B to the final image increases monotonically, depending on the distance and the timer value, until one of the values reaches 1.0 and the information of the blend-in screening mask B of the current fragment is transferred to the screening mask S.

To maintain the screening mask S, an advection according to the velocity value for each pixel is performed every frame. Due to the distortion areas in the velocity field the advected screening mask S contains areas in which the screening matrices are skewed; the method described in the previous paragraph to update B inserts new matrices in the blend-in screening mask B to smoothly replace the distorted matrices in the screening mask S over a couple of frames. Therefore, during the calculation of the screening mask S it is only important to detect when the blend-in screening mask B has been completely blended. Then the screening matrices are copied from the mask B to the screening mask S.

Implementation on Graphics Hardware

Screening masks are stored as textures and applied using a smooth threshold function [74] to evaluate the screening result. Independently, the screening is applied using mask S and mask B in two passes and the results are blended according the maximum of the timer and distance values. During the update of the blend-in screening mask B, a hardware-based backward texture advection [204] is applied to the mask, utilizing dependent texture lookups and maintaining fractional coordinates. The extra information for the timer and the distance counters for each fragment are encoded in free components of the blend-in mask and calculated as described in the previous section during the update process.

For detection of distortion areas the implementation uses thresholds for the velocity ratio below 0.3 to detect divergence and above 1.1 to detect convergence; if either case applies then

the fragment is scheduled for overwrite.

Halftoning Results and Evaluation

Figure 5.15 depicts a rotating pot; this sequence shows that patterns of the screening algorithm follow the texture of the pot while it rotates.

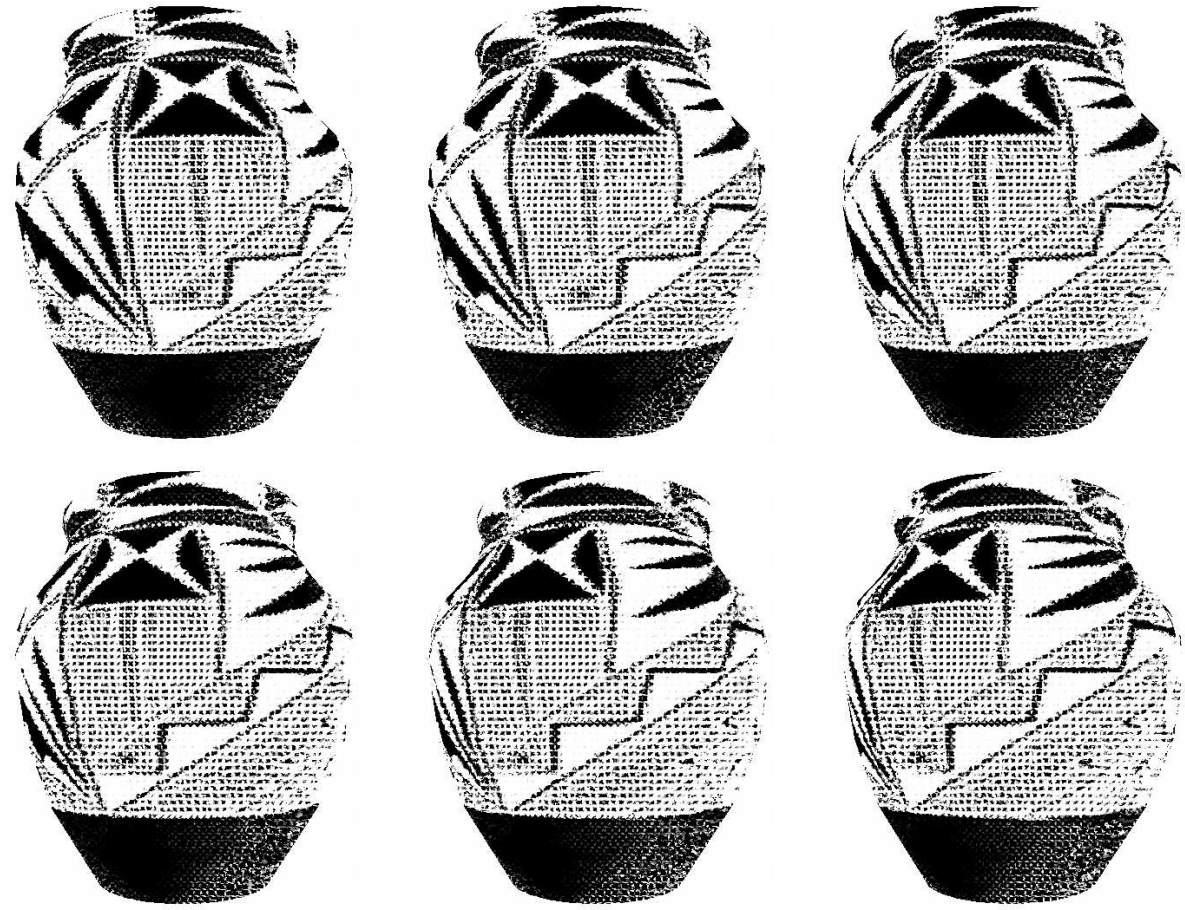


Figure 5.15: Screening illustration of a rotating pot.

Table 5.4 shows the measurement of frame rates in frames per second for different viewport sizes. Frame rates with and without the generation of G-Buffer attributes are displayed; thus, the last row shows the performance numbers for the screening mechanism itself. As test system an AMD Athlon 1.4 Ghz system with an ATI Radeon 9700 PRO graphics adapter was used, the scene used for measuring was a rotating car model shown on the right side of Table 5.4.

The measurements show that even on dated graphics hardware interactive frame rates are achieved. Further, the appearance of the screening patterns is independent of camera zoom, object scale, and object orientation to overcome the shower-door effect even for animated scenes. Since the input to the proposed image-space technique only requires intensity and velocity in

Table 5.4: Frame rates in frames per second.

Viewport size	256x256	512x512	1024x1024
with G-Buffer	238	65	16.4
without G-Buffer	295	68	18.6

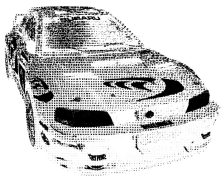


image space, it is also applicable to video sequences. The advantage of halftone renderings on low color-depth displays is illustrated in Figure 5.16.

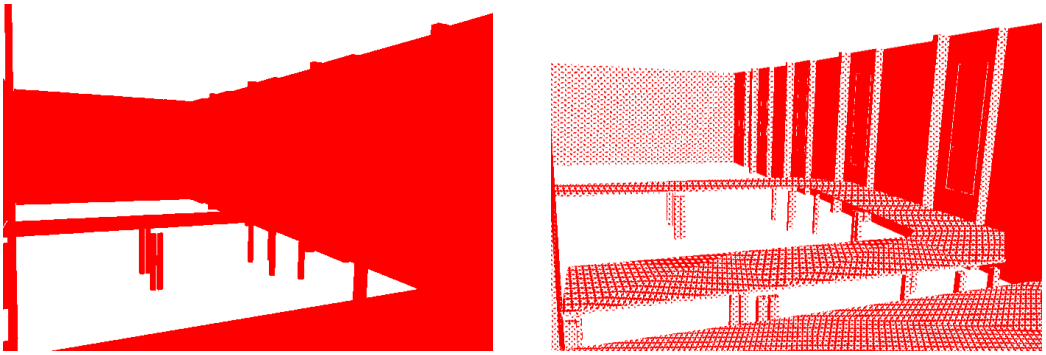


Figure 5.16: A VR scene viewed with a monochrome retina display. Without halftoning (left) only a solid color is displayed. In the simulated screening illustration (right) different intensities are visible due to the applied halftoning.

5.3 Interchangeable Approaches for Data Visualization

Dominant factors in the selection process of adequate visualization techniques are data type, e.g. scalar vs. vector-valued data, and data structure/organization, e.g. regular grids or unstructured data. If datasets differ in these primary attributes, often different classes of visualization methods are required. In contrast, for each combination—e.g. scalar data defined on a regular grid—multiple alternative visualization approaches are know (isosurfaces, direct volume rendering, 3D scatter plots, etc.) termed as *interchangeable approaches*, where each might show benefits dependent on, e.g., the current situation or the considered application. For interactive visualizations any change in these attributes should have a direct effect: an adjustment of parameters—as discussed in Section 5.4—or even a switch to a different visualization technique. This thesis concentrates on alternative visualization techniques for scalar and vector data, but also application-specific data types are considered to show that the basic approach is applicable to other data types as well. In the following sections various alternative visualization techniques for scalar and vector data are discussed to give an impression how context aspects can influence the selection of an adequate visualization.

Although the discussion is limited to specific data types, still not all known techniques to

visualize these data types are mentioned. For both, scalar and vector datasets, numerous techniques for visualization exist and research continuously introduces additional novel techniques. For a basic overview of scientific data visualization, e.g., see [211], for more advanced techniques applicable to various data types C.D. Hansen and C.R. Johnson present a wide collection in [87].

5.3.1 Raw-Value Display versus Visualization

Visualization is the task to display information—most often using a 2D output device. To visually represent multiple data values simultaneously the values have to be placed/projected onto the output's 2D domain. This projection is often based on relations between the individual data elements; most frequently, the elements' intrinsic spatial relations are utilized, if available. In this thesis, such spatial location information per element is interpreted as one aspect of context information; consequently, placement/projection of data values is generically based on context information. Thereby, in the majority of cases spatial location of data values—most datasets explicitly or implicitly define a position for each data element—are naturally remapped to the output domain; if available, timestamps provide further context information that is often considered when presenting temporal datasets.

For datasets for which no natural relation is defined, methods developed in the field of information visualization can be applied to generate an artificial location information (also referred to as layout), often based on attributes of the data; a survey on layout approaches for information visualization can be found in [92]. In the following it is assumed that a location context for each data element is available; furthermore, most shown examples of the proposed context-aware visualizations are targeted for data that intrinsically have a location defined.

The number of data entities and their spatial density (which depends on the previously assigned location information of the data) are both equally important for selection of a visualization technique. For only few, sparse data entities there is often no need for complex visualization techniques and a simple display of raw data values can be used. In the majority of cases this is independent of the underlying data structure or data type, making the display of raw-data values a fallback approach if only few, scattered data elements have to be displayed. Still, it is important to convey context information, e.g. spatial position or tolerance of measured data, so that users can understand the data more easily.

In contrast, if huge datasets or dense data values have to be visualized, individual representation of each value is often inadequate and hard to interpret by humans due to a high cognitive load. In addition, overdraw of annotation—as illustrated in Figure 5.17 (left)—due to limited screen space has to be solved. Therefore, visualization techniques are applied to generate comprehensive representations of the data: E.g., an interpolated scalar filed with color mapping, as seen in Figure 5.17 (right). Note that already the shown contour of the continent provides context information related to the data values that support users in interpreting the data.

This shows that the content of the dataset itself is a further indicator, besides other context attributes, for the selection of a visualization approach so that a context-aware selection approach also has to consider aspects of *content* awareness.

In the following two subsections a collection of alternative visualization approaches for scalar

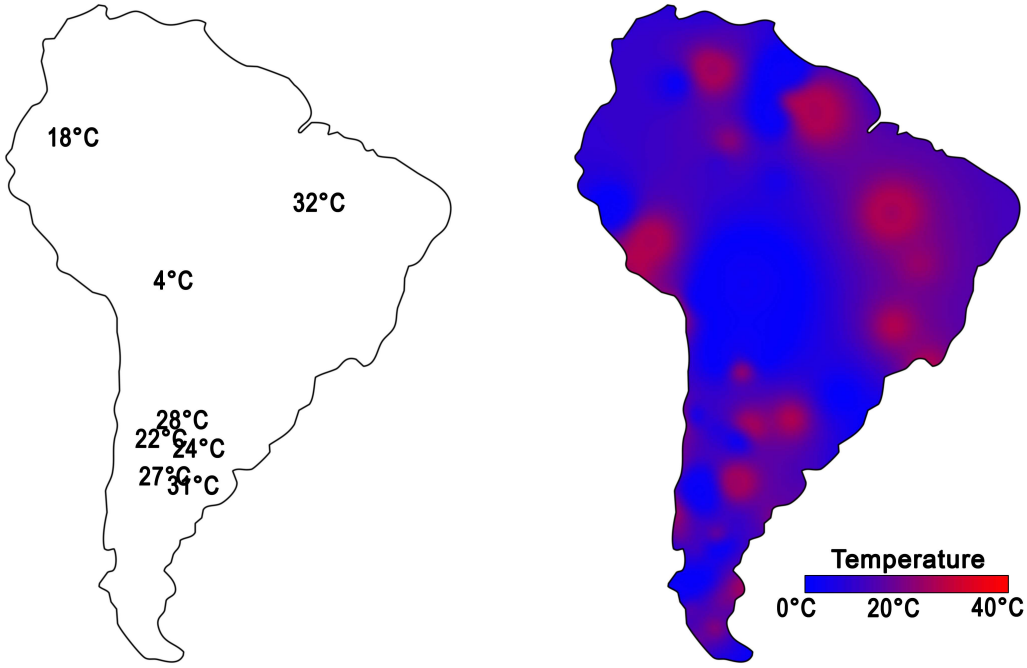


Figure 5.17: Artificial temperature data of South America. When rendering only few raw values users have to manually estimate values in-between (left, upper part). In contrast, if more labels are shown, overdraw occurs (left, lower part). The visualization shown on the right side solves these problems by mapping temperature values to color.

and vector data are discussed with respect to context-aware aspects. Most techniques have advantages and disadvantages that make them more or less suitable for specific scenarios. Each section closes with a table that addresses various context parameters and shows possible visualization approaches. With the availability of such information, the proposed generic framework for context-aware visualizations (Chapter 4) is able to select the most appropriate visualization technique after evaluating relevant context information. Although the presented discussion is limited to only few visualization approaches for two different data types, the diversity is already overwhelming.

5.3.2 Scalar Data

In general, scalar datasets define data values, each with only one scalar component like the primitive data types char, float, double, or integer. Usually, a relation between the individual elements is known, e.g. their location within an n-dimensional coordinate system. Furthermore, within this coordinate system data values may be organized in structures or given as unstructured data. For structured data the connectivity and position is implicitly given by the underlying structure, e.g. Cartesian grids. For unstructured data the connectivity and position has to be stored explicitly for each data element. Clearly, data processing of unstructured data differs from processing of structured data. The data domain defines the region where data values are available;

however, dependent on the semantics of the data—if it represents a continuous phenomenon—an interpolation schema can be applied to determine data values at arbitrary locations within the data domain. Often such a schema is available, e.g. for measured data or scientific simulation datasets, but also data without a defined interpolation scheme is common. If the data is organized using physical context information like a 2D or 3D position, a simple projection can be used to display the data on a 2D screen. For higher-dimensional data specific visualization techniques exist, e.g. the concept of parallel coordinates as proposed by A. Inselberg [99], or a dimensionality reduction method can be applied, e.g. FastMap by C. Faloutsos and K.-I. Lin [68], to project the high-dimensional data to a 3D/2D space.

Scalar data without a defined interpolation often occurs when non-spatial attributes are applied for placement, e.g., purchase price of cars is assigned to x-axis and car type is assigned to y-axis. Independent of the number of cars drawn in such a diagram, values in-between specific car type-price pairs are undefined. In contrast, scalar data which represent time-dependent stock values can be interpolated to estimate a value in-between two known time-value pairs. Simulations and measurements, e.g. magnetic resonance imaging (MRI), often provide structured 3D scalar fields as a representation of the underlying continuous data whereby a 3D location for each measurement point is implicitly known.

Individual Data Representation

In general, visualization techniques are designed according to the visualization pipeline as presented in Section 3.3, thereby each individual data element can be piped through the visualization pipeline separately and combined afterwards in image space to generate the final image or the entire dataset is processed once for all elements, detailed in the following paragraph. A separate processing is often preferred if either the data elements cannot easily be combined, i.e. do not support an interpolation, or if the data items are sparse, i.e. located far apart.

Calculating the visual representation for each data element individually implies that each data element is mapped to a renderable primitive. Naturally, dependent on the amount of input data, this can lead to many graphical primitives that have to be rendered to generate the final visualization result, which probably limits the interactivity of the resulting visualization. To convey a scalar value the representing primitive may show the numerical value; however, numbers are difficult for humans to be interpreted in great amounts, therefore a mapping to color or even icons is preferred. The process of mapping a range of scalar values to a predefined color table is also denoted as *transfer function* and can be used to classify the scalar data. Furthermore, the size of the rendered primitives and the density of the dataset have to be considered in order to select a valid configuration so that overlaps of the primitives are minimized, which would result in undesired artifacts.

Therefore, often only pixel-sized points are draw and the scalar value of each location is expressed by the point's color. Plots constructed in this way are termed scatter plots [88] and are widely used in 2D domain, but seldom for 3D data because of limited depth perception and occlusion problems. If the scalar values can only take few discrete values, icons can be used to express the values, like weather forecasts use icons for cloudy, partially clouded, or sunny weather. To improve the depth perception and alleviate the overdraw problem—especially

of 3D, dense scatter plots—transparency can be used. Rendering semi-transparent primitives offers the possibility to visualize data elements that are otherwise completely overdrawn; this so-called splatting approach can, e.g., be used to render dense volumetric datasets. However, most common blending models require a depth-sorted rendering order, which is in general view dependent and therefore has to be recomputed for each view change, limiting the performance especially for mobile devices. An efficient hierarchical approach to render large point sets is presented by M. Hopf and T. Ertl in [95].

In contrast, if a continuous representation is applicable more sophisticated approaches can be used. Thereby, even scatter plots can be visualized in a continuous way—as presented by S. Bachthaler and D. Weiskopf [8]—to convey the behavior of in-between values.

Aggregated Visualization of Scalar Data

For input datasets with a defined interpolation schema a value can be calculated at each position of the input domain. With the use of interpolation the input data structure can be adapted via so-called resampling, e.g. changing the resolution of the underlying grid or sampling unstructured data to a Cartesian grid. Although resampling might introduce errors—the applied interpolation for reconstructing data values at arbitrary locations is in general not exact—it is a widely applied method to convert input data. Resolution and grid type (including unstructured data) are often adapted to match the limitations of visualization techniques or utilized (mobile) hardware. This allows for some independency of the input data structure and applicable visualization techniques; however, due to the introduced errors visualization techniques that operate on the raw input data structure are preferred.

In contrast to representing each individual data value by one or multiple primitive(s), an aggregated visualization makes use of a proxy geometry covering the relevant data domain. Thereby, the input data is optionally converted, e.g. via resampling, to match this proxy geometry. A common target is a 2D or 3D array which implies a constant data density within the entire domain; Thus, mainly dense data is considered in order to achieve meaningful results.

For representing the values, again the scalar data can be mapped to intensity or color using a transfer function. On the right side of Figure 5.17 a visualization of interpolated scalar data on the relevant domain (South America) can be seen. For visualization the unstructured input data has been resampled to match the image resolution using Shepard Interpolation [185] and the scalar values [0..40] have been mapped to colors from blue to red with a corresponding transfer function.

In real-world scenarios, scalar data is often related to physical objects; therefore context information can be used to include relevant objects in the visualization. Furthermore, the geometry of these objects defines important regions in the data domain and are therefore appropriate to be used as proxy geometry to convey scalar data attributes via, e.g., color mapping. Figure 5.18 presents 2D data within a 3D context: the ground temperature of a section in the city center of Stuttgart, Germany. The context information, namely the surface height field geometry and the orthoimagery data, is used to represent the spatial context of the temperature data and simultaneously serves as proxy geometry on which the scalar data is mapped onto. The presented visualization further makes use of context information to limit the visualization to important re-

Figure 5.18: Continuous representation of 2D scalar temperature values embedded within its real-world context, shown as a textured height field[6].

gions: Only data values that diverge from the average temperature by a certain amount are shown, making the visualization additionally content-aware. Note that the result shows a strict application of embedded visualization (Section 5.1.3) for scalar data; using real-world proxy geometry as visualization canvas has the advantage that context information is not hidden.

A further common approach to present 2D scalar data is to interpret the scalar data as surface height at the corresponding 2D location resulting in a so-called height field representation. Naturally, height fields can be applied to visualize, e.g. measurements of the earth surface via aerial laser scanning [75]. In fact, the visualization shown in Figure 5.18 makes use of such a description to render the underlying terrain height field, which presents the earth surface height as additional context information. If the data values are of discrete nature, i.e. only few scalar values occur in the entire dataset, or boundaries between specific value ranges are examined, e.g. values above 100 m vs. below 100 m, height or isolines can be used to show and emphasize different height regions. Isolines are typically applied in topographic maps to convey different surface heights.

Data values defined in higher dimensional space, namely 3D scalar datasets, that can seldom be mapped onto real-world geometry for display in a meaningful way, require more advanced approaches for visualization. Techniques to interactively visualize volumetric scalar data are still an active field of research. New scanning technology produces higher resolution 3D datasets,

[6]Orthoimagery and height field data is provided by the Institute for Photogrammetry (IFP), University of Stuttgart.

usually defined on a Cartesian grid, but also dense unstructured data is often resampled on regular structures resulting in huge volumetric datasets. Currently, grids of up to 2048^3 with 16 bit precise scalar values are captured with modern scanners (=16 GB), therefore requirements on visualization methods continuously increase. To visualize 3D dense data, again a proxy geometry has to be generated to convey the data. In volume rendering, the discipline of visualizing scalar 3D data fields, two principle techniques are possible: indirect or direct volume rendering.

Indirect volume rendering (IVR) can be seen as the 3D analogon to isolines, termed as isosurfaces. An extraction of one or multiple isosurfaces can either be done offline in a preprocessing step or on demand in real-time during the execution of the visualization method, e.g. using the approach of T. Klein et al. [117]. The most common algorithm for isosurface extraction is marching cubes [129] or marching tetrahedra [52], dependent on the underlying data structure. Similar to isolines, isosurfaces are only applicable to show few discrete values. For scalar data fields with continuous data a direct volume rendering (DVR) approach is more appropriate. Thereby, the scalar data is interpreted as participating media and the visualization is generated by simulating the light transport through it.

In the previous paragraph a splatting approach is mentioned, which is capable of presenting 3D scattered data by projecting each data element on the screen individually. For huge datasets the separate processing of each element tends to result in unacceptable performance. Therefore, an approach that handles all or at least several data elements at once is beneficial. Again, research came up with many solutions that address this specific problem of generating a so-called direct volume rendering visualization. The basic idea behind most approaches is to interpret the data as semi-transparent 3D region with given (local) density values and perform a physically motivated simulation to evaluate the light transport through this volumetric dataset. The resulting realistic image is then used as visualization. Often, the simulation of the light transport is performed by solving the volume rendering integral, as defined by J.T. Kajiya [108] and detailed in Equation 5.14.

$$\begin{aligned} I(s) &= I(s_o)\mathrm{e}^{-\tau(s_0,s)} + \int_{s_0}^{s} q(s')e^{-\tau(s',s)}\mathrm{d}s' \\ \tau(a,b) &= \int_{a}^{b} \kappa(s')\mathrm{d}s' \end{aligned} \tag{5.14}$$

The equation calculates the accumulated intensity I at position s along a ray through the volume dataset, starting at s_0. The emission of light at a given location x within the scalar volume is defined by $q(x)$; absorption is denoted by $\kappa(x)$. Note that Equation 5.14 calculates the light transport in a volumetric media without considering scattering or frequency effects. Although the volume integral does not consider all effects, an evaluation for interactive applications is still very costly, so that GPU-based approaches are proposed; the most common techniques are: 2D slice-based, 3D texture based [40, 210], and shader-based raycasting [189]. A 3D texture-based approach is implemented as a context-aware visualization and detailed in the following.

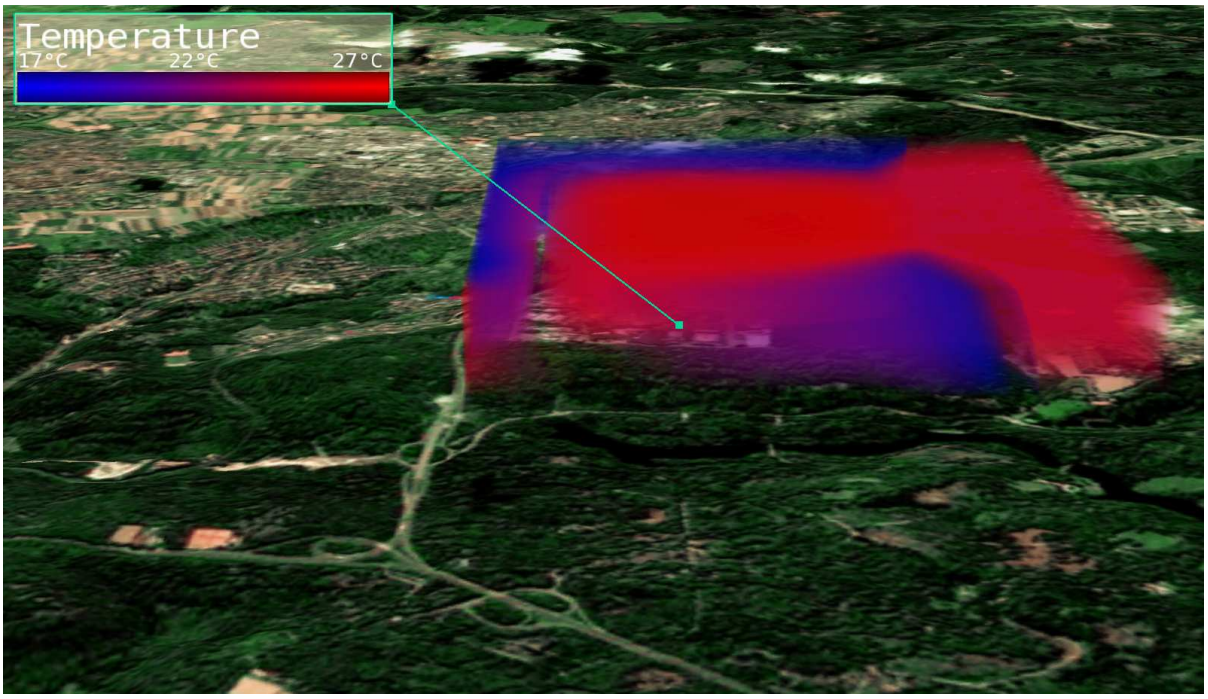


Figure 5.19: 3D scalar data visualized using a volume rendering technique to allow a continuous display of scalar values within the entire data domain[7].

Rendering 3D scalar data is achieved by generating geometry of view-aligned slices though the 3D dataset. These view-dependent slices are rendered in a back-to-front manner with a 3D texture applied that stores the data values. During rendering the slices, blending is activated to accumulate the values for each pixel. To classify the scalar data values of the dataset a transfer function is applied that maps each data value to a defined transparency and optionally to a color, which both influence the accumulation and, thus, the final image. After rendering all slices, the visualization is complete as seen for a 3D temperature field in Figure 5.19. Note that the data is not only shown for one layer but for a volumetric region compared to the 2D temperature filed in Figure 5.18.

As mentioned, IVR is preferred if only few specific data values are considered whereas DVR presents the entire range of data. An context-aware automatic approach to decide which technique to apply can make use of a histogram analysis. If the histogram shows only few peeks over the entire value range, an IVR approach can be used, otherwise a DVR method is more appropriate.

The following matrix in Table 5.5 considers a small excerpt of different context configurations and shows possible, corresponding visualization techniques that are adequate for scalar data. This matrix does by no means claim to be exhaustive. For visualization, location information for each data values is required; most often, data values already provide 2D or 3D location

[7]Orthoimagery and height field data is provided by the Institute for Photogrammetry (IFP), University of Stuttgart.

information, otherwise additional context information might be used as location. Table 5.5 shows the location context on the top row, as location is a major aspect of the context; location information is separated into 2D, 3D, or nD data. Dependent on the dataset, the values might be sparsely or dense. For simplicity, dense and regular datasets are combined as dense data can be resampled to regular grids introducing only small errors. Discrete data has only few values/states and can therefore not be interpolated, as values in-between are invalid. Additionally considered context attributes, besides location, are listed in the left column of Table 5.5. Although multiple aspects might be valid concurrently, only combinations with the location context are laid out.

Most techniques suggested in the table are well-known approaches and can easily be found in basic visualization literature [211, 87]. An extension to scatterplots for spatially continuous data is offered by continuous scatterplots as proposed by S. Bachthaler and D. Weiskopf [8]. Alpha-shapes provide a generalization of the convex hull as mentioned by H. Edelsbrunner and E.P. Mücke in [53]. Various focus&context approaches, e.g. blurred scatterplots, are discussed by R. Kosara in [121]. The application of focus&context in volume rendering is, among others, presented by J. Krüger et al. with the application *ClearView* in [124]. Pre-integrated volume rendering is an efficient technique for high-quality volume rendering [65]. Linked 2D/3D scatterplots are presented by H. Piringer et al. [161]. P.J. Rhodes et al. map data values to intensity and respectively their uncertainties to hue for the visualization of isosurfaces [169]. Again, these are only examples how various context aspects can effectively be used to control the selection of adequate visualization techniques, where ongoing research will most certainly further expand the matrix presented in Table 5.5.

In the following a similar discussion is presented considering vector-valued data; although scalar and vector data are separated here, the proposed framework for context-aware visualization conceptually makes use of a single, joint matrix where all relevant aspects are considered, i.e. data type is a further attribute of the considered context.

Table 5.5: Visualization techniques for scalar data in respect of various context aspects.

	context aspects mapped to location (if available typically the spatial location of the data value)						
	mapping further context attributes (e.g. time) →			projection/dimension reduction ←			
	2D			3D			nD
additionally available context aspects	sparse discrete	sparse continuous (resampling →)	dense/regular continuous	sparse discrete	sparse continuous (resampling →)	dense/regular continuous	
considered value range							
dynamic/full	scatterplot	continuous scatterplot	color mapping	3D scatterplot	splatting	volume rendering(DVR)	scatterplot matrix
discrete/few	placed icons or glyphs	convex hull, alpha shape	isolines, contours	glyphs in 3D		isosurface(IVR)	
description of importance							
viewing distance (to POI)					hierarchical splatting	focus&context DVR, ClearView	
bounding volume	focus&context scatterplot			focus&context 3D scatterplot		DVR with clipping	
related geometry	value pointers		color-mapped geometry			color-mapped geometry	
quality of data							
accurate/high quality	text labels	natural neighbor		linked 2/3D scatterplots		pre-integrated DVR	parallel coordinates
uncertain/low-quality	per value error bars	intensity vs. hue mapping		uncertainty glyphs		uncertainty isosurfaces	p. coord. & probability

5.3.3 Vector Data

Similar as for scalar data, there are probably even more visualization methods available for vector data. A numerical display of vector values is even less appropriate as multiple values have to be displayed per data element. Furthermore, an interpretation of numerical vector values is often not easy, thus a different approach to present the data is required; e.g. it is more appropriate to draw an arrow to express a vector value. Instead of visualizing the entire vector value, a derived scalar value can be visualized using one of the aforementioned techniques for scalar data. For an overview of the entire dataset already a scalar value might show regions of interest which have to be investigated further, preferably using a more sophisticated visualization approach that is adequate for vector data. A common example for visualizing interesting regions within a flow field (a field where vectors describe the direction and speed of flow) is to calculate the magnitude of the velocity at each location and visualize the resulting scalar field. If a reduction of the data to scalar values is inappropriate then techniques have to be used that are able to achieve more meaningful representations.

Individual Data Representation

Other than scalars, vectors are often interpreted as arrows showing a direction and a magnitude via the arrow's length. Drawing an arrow for each vector-data entity at its corresponding position can be used for vector visualization; note that presenting the numerical raw data values of a single vector is already hard to interpret for users. However, representing each data entity separately quickly leads to an overdraw problem. Borrowing ideas from scalar visualization approaches allows to alleviate the overdraw problem by drawing arrows with equal length to convey direction and applying a transfer function to map magnitude of vectors to color. Still the expressiveness of such a simple method is limited and seldom applicable for end users, but it can be used for debugging during algorithm development. In fact, Figure 5.20 shows a debug output example of a 2D vector field representing a flow field of a fluid simulation that is intended to drive a realistic fluid-surface render proposed together with T. Klein and D. Weiskopf in [116].

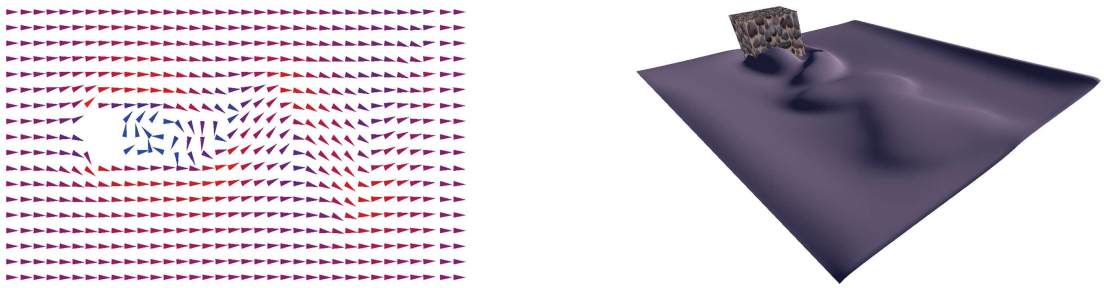


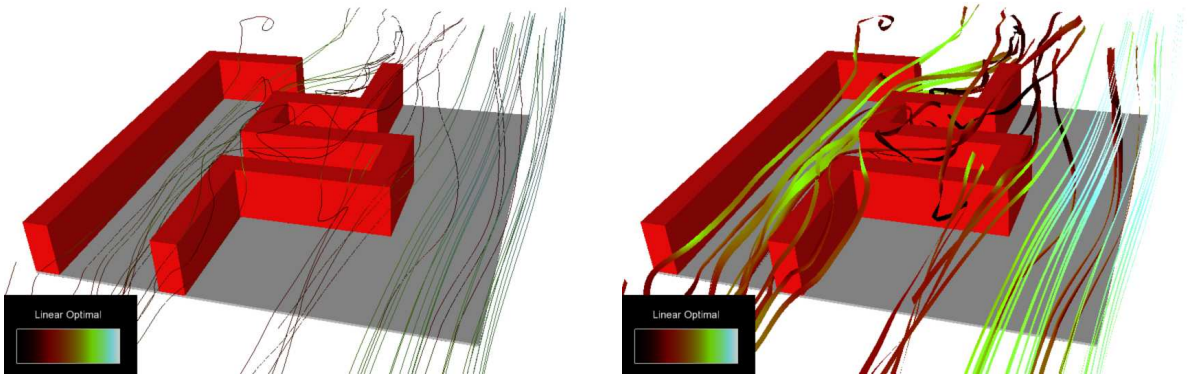
Figure 5.20: Simple visualization of a 2D vector field (left) used as debug output during algorithm development for realistic fluid behavior (right) [116].

In order to understand and interpret flow behavior, more sophisticated approaches are required that show additional important characteristics like curl within 3D flow fields.

Aggregated Visualization of Vector Data

Typical vector data is not only defined at discrete position but represents a continuous field of data, i.e. the data might be measured/simulated only at selected locations (usually a grid) but an interpolation is defined to evaluate values at arbitrary locations. Therefore, techniques that calculate vector-field visualizations usually do not process data entities individually but generate a representation of the vector field for the entire data domain.

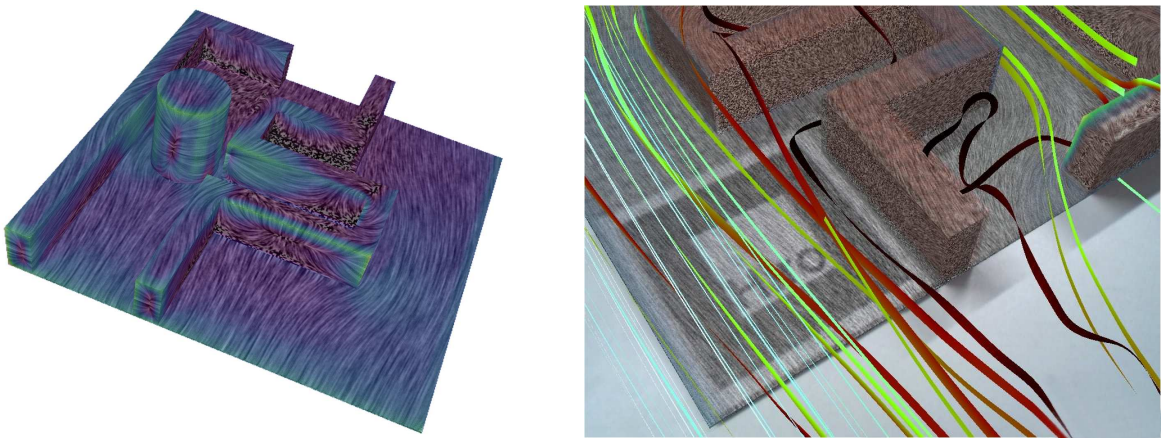
For visualizing basic aspects of a vector field, independent of possibly present geometry information of the physical context which might be used to convey highly detailed information, again an introduction of proxy geometry for visualization is required. Streamlines are a well-known basic approach to visualize flow direction and give users a good overview of the global flow behavior. The lines are generated by numerically calculating positions by following massless particles inside the vector field over time. Separately, for each particle its positions are connected to generate a streamline. Additional scalar information, e.g. magnitude of the velocity or pressure, can be visualized by color coding the streamline geometry. Figure 5.21a shows a flow overview in context with the geometry of the scenario; color mapping is used to show the magnitude of the flow's velocity on the streamlines.



a) b)

Figure 5.21: Visualization of a flow-field overview using streamlines (a) and stream ribbons (b) embedded within the geometrical context of the scenario. The lines are used to present the flow direction whereas the magnitude of the velocity is illustrated by the color coding.

Streamlines have the advantage that their thin structure is appropriate to embed them even within geometrically complex scenarios. However, the thin lines also have disadvantages: 1D objects cannot provide a good depth impression, which is very important for spatial perception, curl of the flow is not expressed, and the perception of the color coding is limited due to the thinness of lines. The limited depth impression can be improved with illuminated streamlines according to a method described by Zöckler et al. [213]. This helps to get a better depth impression and makes it easier for users to interpret the flow data. Also halos can be added which enhance the contrast to the scenario's context (background). This is especially important if a real-world context is provided via a video stream, like in most augmented reality setups.



a) b)

Figure 5.22: Vector field visualization using line integral convolution. Flow direction on surfaces and the magnitude of velocity are illustrated in a). Further, b) adds stream ribbons to present the flow behavior in geometry-free regions.

Beside these enhancements, streamlines can be extruded to 2D stream ribbons. Ribbons are 2D objects and therefore provide inherently a better depth impression than streamlines and are able to convey the curl of the underlying flow. Figure 5.21b shows a visualization using stream ribbons, it can clearly be seen how the curl of the flow twists the ribbons. Furthermore, the enlarged geometry improves the perception of the color mapping, but thereby, more screen space is occupied by the ribbons, hiding important context. Therefore, streamlines might be preferred in situations where the context is most important.

In general, both line-based approaches are appropriate to present a global overview of the vector-field structure but fail to show small, detailed structures. Visualization techniques that present fine structures of the flow—like line integral convolution (LIC)—can be used to present flow orientation using an embedded visualization approach directly on the scene geometry, similar to color mapped surfaces. Such a dense flow visualization method is proposed by D. Weiskopf and T. Ertl [205]. They describe an interactive LIC technique for arbitrary geometry so that the flow features can be presented directly using the geometry context of the scenario. The texture-based visualization provides a detailed impression of the flow movement in every point on the surface. The main idea of their proposed method is to combine an image-space method with some computations performed in object-space. This hybrid method can efficiently be implemented on graphics processing units (GPU), which results in highly interactive rendering performance, a strong requirement for context-aware visualization. As LIC techniques present flow direction by exploiting the contrast of a noise texture, further attributes of the vector field can be mapped to color. Figure 5.22a shows a LIC visualization that was generated according the technique described in [205]. As depicted, the geometry of the scenario, provided as context information, is textured to present the flow direction and at the same time color mapping is used to express the magnitude of velocity.

Due to simple overlay onto physical surfaces this embedded visualization maintains impor-

tant aspects of the context—the geometry of the underlying physical scenario is still presented—making the approach especially applicable for AR applications. In addition, combinations with other flow visualization techniques, e.g. stream ribbons, can easily be achieved since they do not interfere with each other. A combined visualization is used in the prototype proposed in [57] as shown in Figure 5.22b. Volumetric 3D LIC as proposed by M. Falk and D. Weiskopf [67] tries to visualize fine 3D flow structures by extending 2D LIC to 3D. However, extensive occlusion of flow features and the contextual scenery makes 3D LIC often inappropriate for combined visualizations where context plays an important role.

In extension to the previously discussed scalar visualization approaches, the following matrix in Table 5.6 shows several context aspects and their related appropriate visualization technique for vector-valued data. Again, the considered techniques and context aspects are only a small selection of available constellations.

Basic literature on visualization techniques [211, 87] discusses most techniques that are named in the table. Stream surfaces are an extension to streamlines and can be calculated in Cartesian, but also in tetrahedral grids as proposed by G. Scheuermann [178]. If only the global behavior of a vector field is important, a topological analysis can be applied to simplify the visualization [202], which is even possible for irregular girds as proposed by X. Tricoche in [194]. To adjust the presented detail of the vector field, level-of-detail concepts are available for various approaches as, e.g., for streamlines [105]. In general, many extensions to well-known techniques improve the resulting visualization quality and thus its usability, like the approach for evenly-spaced streamlines presented by Z. Liu [128] or the utilization of more complex streamline geometry to generate streamtubes, to provide an improved depth impression. Furthermore, T. Delmarcelle and L. Hesselink propose hyperstreamlines to visualize tensor fields [43], which can be applied to vector fields to present additional aspects. Even illustrative techniques can be applied to represent multiple time steps in a single image as shown by W. H. Hsu et al. [97]. The handling of inaccurate or uncertain vector data is also considered in visualization approaches and presented, e.g., for 2D flow data by R. Botchen in [24]. This variety of available methods for vector field visualization shows that it is even more important to evaluate context information for selection of appropriate techniques, as the diversity greatly exceeds the previously discussed scalar-data case: ranging from simple per-element arrows up to high-level topological flow-field analysis visualization, e.g. proposed by T. Weinkauf et al. [202].

Up to now, data types that are common in scientific visualization were covered, namely scalar and vector data, but often applications are required to operate on specific data formats and types that cannot be easily described in such generic formats.

5.3.4 Application Specific Heterogeneous Data

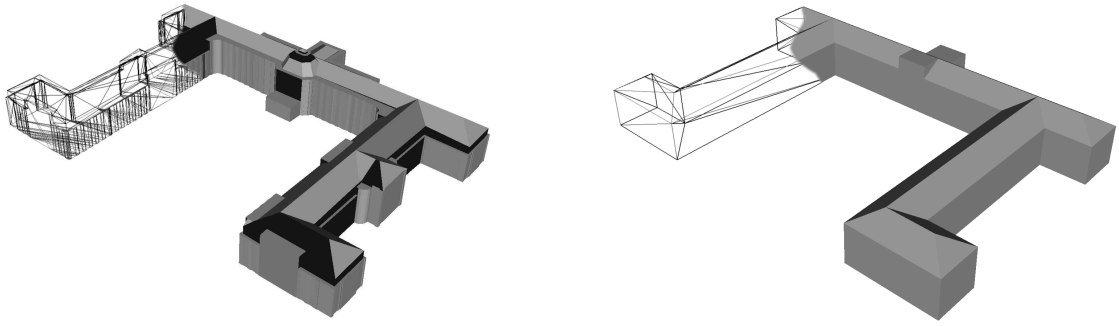
Aside from popular data structures like scalar or vector fields, application specific data is frequently required to be visualized; as no restrictions are placed on the data many custom visualization approaches appear. Nevertheless, usually multiple concepts to visualize such specific data are possible, most having advantages and disadvantages depending on the intended purpose.

In joint preliminary work with M. Peter of the Institute of Photogrammetry at the University of Stuttgart, the quality of a generalization approach for buildings of virtual city models is

Table 5.6: Visualization techniques for vector data in respect of various context aspects.

additionally available context aspects	context aspects mapped to location (if available typically the spatial location of the data value)				
	mapping further context attributes (e.g. time) →		← projection/dimension reduction		
	2D		3D		nD
	resampling →		resampling →		
	sparse/unstructured continuous	dense/regular continuous	sparse/unstructured continuous	dense/regular continuous	(*often: 3D + time*)
considered value range					
dynamic/full		line-integral convolution (LIC)	tetrahedra-based stream surfaces	streamlines	path-/time-/streak-lines or -surfaces
discrete/few	topology simplification	topological analysis		vortex extraction	
description of importance					
viewing distance (to POI)		level-of-detail streamlines		streamlines vs. stream ribbons	
importance function (DOI)		controlled seeding of streamlines		output-sensitive 3D LIC	
related geometry				LIC on arbitrary surfaces	
quality of data					
accurate/high quality	per-value arrows	evenly spaced streamlines		streamtubes / (hyperstreamlines)	illustrative time evolvement
uncertain/low-quality	animated arrows	uncertainty flow visualization	uncertainty glyphs		

visualized. Generalization of 3D building models removes details from models to reduce their complexity; an example is seen in Figure 5.23.



a) b)

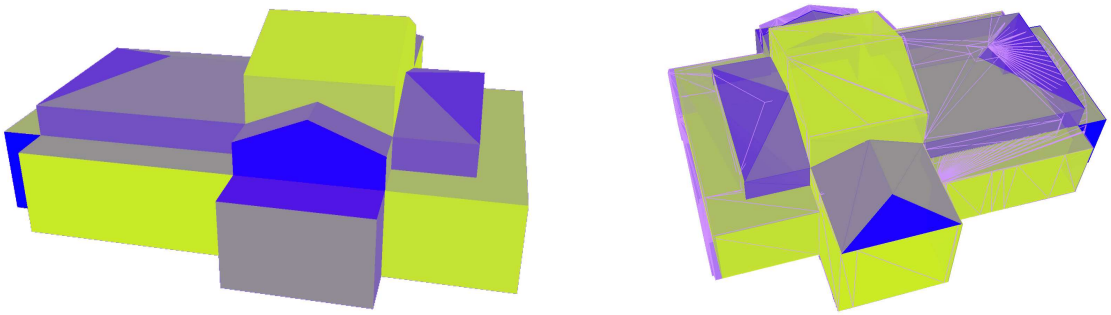
Figure 5.23: Generalization/simplification of complex geometry models; a) shows the original complex model, b) is a simplified low-detail version.

M. Peter et al. propose methods to minimize inconsistencies of simplified 3D building models in relation to their context, e.g. the fixed ground plan or important facade lines of neighboring buildings [160]. The following prototype visualizations are based on data generated by their system. A detailed discussion on the algorithm and how the consistency between 3D models is evaluated is given in [159]; values considered in the visualization are per-surface normalized differences of generalized vs. detailed models: surface area, surface orientation, and surface distance. The data further provides a combined scalar error value representing a weighted sum of the aforementioned differences per surface.

A visualization of this data helps to show the resulting quality consistency of the simplified and corrected data in relation to the original detailed data. Clearly, the source data is not in a common structure—like scalar or vector fields—and is therefore termed as application specific. A common approach to develop first visualization concepts is to convert/map the data to one or multiple existing visualization implementations. Figure 5.24a simply maps the combined scalar error value to a green-to-blue transfer function and colors the surfaces of the simplified model accordingly.

Furthermore, to support users in understanding and probably improving the generalization process, the generated illustrations should show the original data for comparison. This context information helps to estimate the impact of an error. However, showing both models for comparison leads to occlusion problems, therefore the visualization in Figure 5.24b displays only important aspects of the context via the wireframe geometry of the original building model. The resulting visualization shows that for surfaces with higher error value (shown in blue) the original model's geometry often differs significantly: in the upper right the wireframe indicates a roof part with a round structure whereas the simplified version only shows a planar structure.

Such techniques are simple and can easily be rendered at interactive frame rates, even on low-performance mobile devices. Thus, a consistency overview—if multiple possibly huge models are examined—to give users indications of potential inconsistencies can be generated using these techniques. Although the visualization presents some aspects of the data, still users are unable to



a) b)

Figure 5.24: Consistency of a generalized building model. A per-surface scalar error value is used to color the surfaces in a). In addition, the visualization can be combined with the wireframe geometry of the original complex model as shown in b).

investigate the reason of errors, i.e. which of the aforementioned values caused the high combined scalar error. Thus, for a detailed analysis of the consistency, a different approach is required in order to display the raw values on which the algorithm works.

In a sense, the data represents multivariate data where multiple independent data values are given per location. Glyphs are one concept to visualize such data: simple icon-like representations are used to convey different attributes. Figure 5.25 shows two extremes of a simple glyph that illustrates the difference in three attributes between two surfaces (original surface vs. generalized surface). The normalized attributes are additionally mapped to a green-to-blue transfer function to support users in estimating the shown glyph values, i.e. minimal difference is shown green and maximal difference is presented in blue.

Figure 5.26 shows the resulting visualization when using glyphs to visualize the raw data values per-surface. As the raw values are already shown by glyphs, the surfaces are colored differently: at each vertex, the combined per-surface error value of adjacent surfaces is averaged. Thus, regions and vertices that cause inconsistency on multiple surfaces are emphasized.

Note that glyphs are always vertically aligned, independent of the virtual camera. A visualization as shown in Figure 5.26 runs interactively on desktop computers, but on mobile devices the performance may already be too slow. Primarily the number of primitives is the limiting factor, in the used prototypical implementation each glyph consists of over 100 triangles. Although their tesselation level could be reduced—in extreme a block and a rectangle could be used to draw a glyph—still the number of triangles required to draw a glyph would be too high. A possible solution to overcome this limitation is to trade image quality for performance. By pre-rendering a fixed number of glyph configurations into textures, the interactive visualization only has to choose the best matching configuration and draw a textured sprite at the corresponding location in image space. However, due to the visual complexity of the resulting images, an application of glyphs should only be used for close-up views to expose all details of the raw data.

These preliminary results show how basic techniques can be selected based on context aspects and applied to visualize heterogeneous data. Further examples that make use of context information to embed consistency visualizations within presentations of related context are pre-

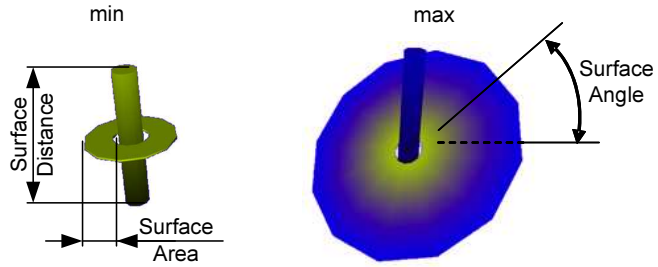


Figure 5.25: A per-surface consistency glyph. A normalized difference between two surfaces in three attributes is presented: distance, area, and angle (orientation).

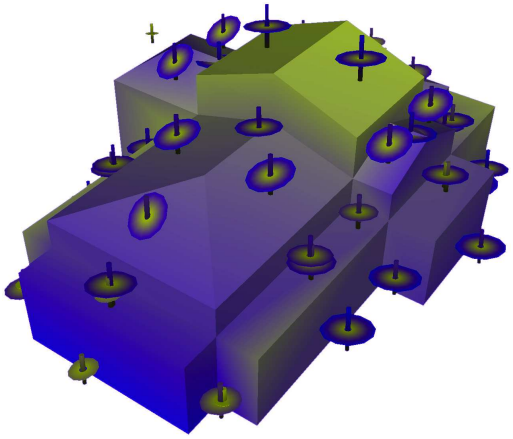


Figure 5.26: Consistency visualization on the generalized model of Stuttgart's opera. Per-surface glyphs are used to present multiple aspects that cause inconsistency.

sented in Section 6.5.1.

5.4 Configuration and Adjustment of Techniques

In the previous section alternative techniques to visualize specific data were discussed. Dependent on context information, some techniques are advantageous vs. others, so that an evaluation based on context information allows an automatic selection of the most appropriate visualization technique. In contrast to choosing entirely different techniques, an extension to this approach is achieved by fine-grained context-aware configuration and adjustment of the chosen techniques themselves.

Section 4.3 details how the proposed framework supports dynamic adjustment of visualization according to context attributes. Thereby, an update mechanism communicates a description of the current context to the visualization which then reconfigures itself, based on this information. Most visualization algorithms have parameters to adjust various aspects, e.g. color table, geometrical detail, accuracy, seed-point locations, etc. Often, context information can be evaluated to adjust these attributes adequately. For some of the aforementioned visualization techniques possibilities of context-aware adjustments are highlighted.

5.4.1 Context-Aware Streamlines and Stream Ribbons

Flow visualization using streamlines or stream ribbons introduce virtual objects representing the flow direction. Thereby, the objects should preferably be placed in interesting regions, e.g. only few lines have to be drawn in regions of laminar flow. In case of streamlines and stream ribbons it is often up to the user to define so-called seed points that spawn lines, e.g. by defining

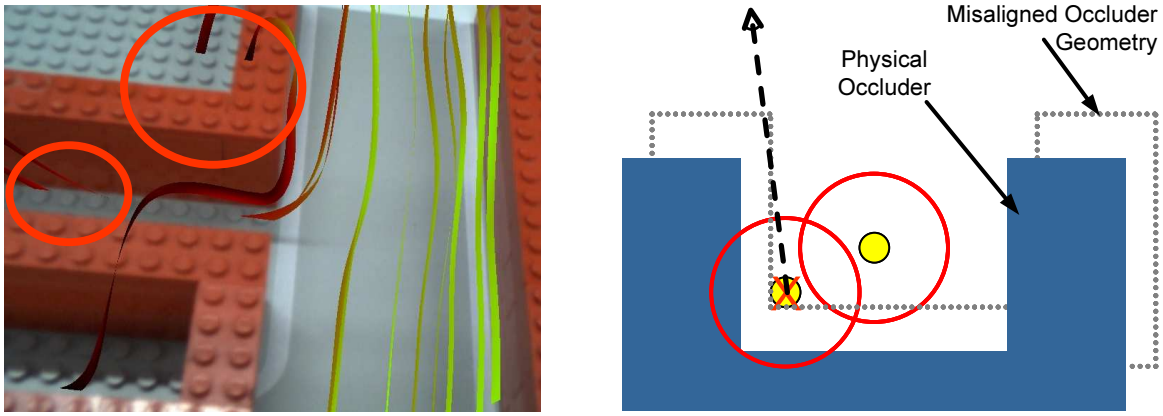
a region in which a certain number of seed points are evenly distributed. Two main problems arise from this user-oriented approach: First, users are usually unaware of interesting, important regions beforehand, i.e. before a visualization of the flow is available. Second, for mobile or VR/AR applications the possibilities for advanced user interfaces are often very limited and user interaction should be minimized. Therefore, VR/AR scenarios often implement interaction via physical manipulation—like changing the camera orientation by head movement.

In order to make the usage of streamlines more practical in these systems, an automatic placement of seed points is preferable. To achieve this, a statistical method for seed-point placement described by M. Zöckler et al. [213] is used. A scalar value p_i describing the importance of the flow at a position i is used for the decision where to place a seed point. M. Zöckler et al. suggest a content-aware approach by using the magnitude of the velocity or any other aspects of the flow data that characterize interesting regions. With the availability of more general context information, not only the flow data itself, any other influencing aspect of the context can be used to build the scalar relevance field. Candidates are the field of view and position in focus, animated and static geometry of the scenario, or screen resolution of the output device. Furthermore, often a single attribute cannot adequately control the placement; when using the flow's velocity magnitude for p, it could happen that many seed points are placed far away from physical objects embedded within the domain of the vector field, because there are regions of higher velocity due to absence of obstacles, but often the behavior of the flow near physical surfaces is important. In a prototypical implementation of a context-controlled flow visualization [57] the scalar values p located near physical objects are be multiplied with a constant to avoid too many seed points that are far away from real objects. With this approach geometric context information is used to generate seed points for streamline integration that are primarily placed in regions of physical obstacles and high magnitude of velocity.

For generating the final seed points, based on the relevance function p, points are distributed randomly, where the probability that a seed point is placed at a position i is proportional to the scalar value p_i. The illustrations in Figure 5.21 make use of this placement strategy to find adequate seed points automatically. In addition to the placement, the total number of streamlines/-ribbons also has a great influence on the final visualization. If too many lines are rendered, the context of the scenario is covered or even aspects of the flow are obscured due to self-occlusion; using only few lines might also hide important details of the flow. Therefore, a tradeoff between both extremes has to be found. Again, context information helps to control this parameter. A behavior in which few lines are used to visualize an overview of the flow field for distant views, and additional lines or ribbons are added in if the view gets closer to reveal details of the flow can easily be modeled utilizing the context information of the current view configuration.

Dependent on the scenario, specific context aspects might have a more dominant controlling effect compared to others. In the work presented together with M. Kreiser [57] an augmented reality visualization is dynamically controlled, based on the quality of the currently available position and orientation information. Position and orientation information—also referred to as tracking data—in AR application have to be highly accurate, which is a hard to fulfill requirement in most practical applications. With the use of context information about the currently observed tracking quality, the visualization can adapt itself to react upon reduced quality of position or orientation information.

The proposed system utilizes this information to alleviate the effects of occlusion due to misaligned geometry, which are depicted in Figure 5.27a. As typical for interactive AR applications, a virtual model of the physical scenario's geometry is used to solve occlusion of virtual objects—e.g. streamlines, glyphs, etc.—with real-world geometry (see Section 4.3.4). Users are often distracted by such noticeable alignment errors of virtual vs. physical world, although the underlying data might have an even lower accuracy. In fact, the simulation to evaluate the flow vector field that is used in Figure 5.27a has a resolution of roughly one Lego peg. Therefore, a misaligned visualization does not necessarily show false information.



a) b)

Figure 5.27: Illustration a) shows false occlusions of streamlines due to misalignment of virtual occluder geometry vs. physical geometry. By showing only lines with a certain distance to the real-world geometry, the number of wrongly clipped lines are reduced, as depicted in b).

In order to diminish occlusion artifacts, streamlines/-ribbons are identified that most likely cause occlusion artifacts regarding the currently observed tracking quality. In general, occlusion of objects is strongly dependent on view parameters, i.e. position and direction of the camera. It can further be observed that independent of the view configuration, geometry close to occluding objects is more often (partially) occluded than geometry at greater distance to occluders (see Figure 5.27b). Therefore, the probability that a streamline is partially occluded by geometry of the physical scenario increases if the distance of the streamline to the real-world geometry is decreased. The proposed context-aware adjustment of streamlines makes use of this fact by presenting only lines or ribbons that preserve a certain distance to the—possibly misaligned—occluder geometry, used to simulate the occlusion of virtual geometry by physical objects (see Section 4.3.4).

A possible solution for an efficient implementation, also used in the prototype, is to sort generated streamlines into bins dependent on their distance to the scenario's real-world geometry. A simple sample-and-reject algorithm is used to fill the bins according to a given distribution function. If a streamline is generated during preprocessing and the corresponding distance bin is already filled, the line is rejected and another seed point for a new line is calculated. The maximum number of retries is limited by a user-defined threshold to ensure termination of the algorithm. For the prototypical implementation, a linear distribution function for the streamlines

was used, so that the largest bin holds the lines with largest distances and the bin with lines that have only minimal distance hold only few lines. The visualization method activates bins of streamlines that should be rendered based on the currently observed accuracy of the position tracking, i.e. the quality of context information, to accomplish that misalignments are less noticeable.

As mentioned above, occlusion depends on the viewing position and direction which motivates a view-dependent calculation of the minimal applicable distance of stream ribbons/-lines to occluder geometry. Furthermore, if the exact error of the tracking system is known, the maximum translation and rotation error can be calculated and used to evaluate the correct minimal allowed distance of streamlines/-ribbons to the scenario's real-world geometry. However, both approaches will introduce a temporal incoherence, i.e. in-/out popping lines, even for minimal changes of the view orientation. For interactive applications temporal coherence of the visualization is important since users continuously adjust parameters while working, whereby strong popping artifacts would confuse users. Therefore, a simple empirically developed mapping table of tracking quality to minimal allowed distance was used.

How context aspects can be used to select an appropriate visualization technique has been discussed in Section 5.3.3. However, often the advantages and disadvantages for different visualization techniques do not clearly identify the most beneficial method, e.g. streamlines (minimal occlusion of context) vs. stream ribbons (revealing the flow's curl). For specific configurations a continuous transition between visualization techniques is possible: For an overview of the flow field, at distant views, streamlines are used that are successively extruded to stream ribbons—in a temporal coherent manner—according to an approaching view point. In general, such a smooth transition between visualization approaches can hardly be achieved. Alternatively, combined visualizations—i.e. complex visualizations that are composed of multiple techniques presenting the same or different information—can be used where only some parts of the entire visualization are changed at a time to maintain temporal coherence. When combining multiple visualization techniques, their mutual influence have to be considered and minimized. For the examined scenario of streamlines and stream ribbons the relevance function p, which controls the placement, can be adjusted by context data to remove streamlines/-ribbons from spatial regions where another technique is used to convey the vector-field data.

5.4.2 Context-Controlled Application Specific Visualization

As an example for visualization of application specific data types, different preliminary approaches were described in Section 5.3.4. In the following concepts for a fine-grained dynamic adjustment of these techniques based on context information are presented.

The combined visualization of a generalized building model and its original complex model rendered as a wireframe overlay, as shown in Figure 5.24b, gives a good impression of the applied generalization approach. However, if the original model is highly complex, then several problems arise, dependent on the context. On low-performance devices—possibly without GPU support—rendering of many lines at interactive frame rates is a non-trivial task. Therefore, only significant lines should be rendered in regions where the generalized and the complex model differ most. Furthermore, the physical display size and its resolution on mobile devices often

cannot adequately show many lines that are only marginal apart, resulting in lines that merge to a solid-colored area, especially if the visualization shows a zoomed-out view of the entire model. Again, rendering only important edges helps to reduce the total number of drawn lines, an approach often applied in non-photorealistic rendering methods [174]. In particular for the considered example, simple geometry that is generated by extruding the building floor plan can be used, which dramatically reduces the number of lines as possibly complex roof structures are neglected. Additionally, a level-of-detail mechanism could be applied: controlled via view parameters, details with many lines are shown for close-up views; whereas only few lines are rendered for zoomed-out views.

When using a glyph-based visualization, performance improvements for limited client devices are already mentioned in Section 5.3.4. Using pre-rendered glyph images reduces the required computation power, but still further improvements are advisable. Especially mobile devices with small screen displays can only present few glyphs at an adequate size, in order to be perceptible by users. A natural solution of this problem is to reduce the number of displayed glyphs, either dependent on the view configuration or by placing an upper limit to the total number of shown glyphs. In order to remove glyphs, a hierarchy is introduced to merge multiple, spatially nearby glyphs so that important information is still presented but in a condensed way. Thereby, higher-level glyphs represent the maximum error of its children in a way that the maximum of each of the three values a glyph represents (see Figure 5.25) are individually selected. Figure 5.28 shows the result of a single merge operation to generate a higher-level glyph.

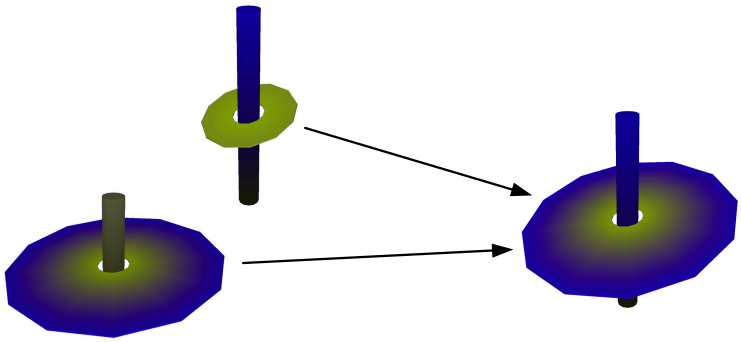


Figure 5.28: Merging multiple consistency glyphs to reduce the total number of glyphs in the visualization. The merging operation maintains the maximum error of each aspect the original glyphs represent.

With the availability of higher-level glyphs the visualization approach can dynamically adjust the number of shown glyphs based on various context aspects like total number of shown glyphs, client hardware performance, view distance, etc. Even focus&context methods, as introduced in Section 5.1, can be applied to show per-surface glyphs in a visualization's focus area and consolidated glyphs in the surrounding context area.

5.5 Open Issues for Context-Aware Visualization

Considering context data to control applications' behavior is already an often discussed topic in research and will most certainly become even more important, particularly in research on data visualization. The designs of novel visualization approaches are often motivated by specific aspects of a considered scenario, exactly these characteristics are important for a context-aware behavior since the new approaches work best under these conditions. Therefore, researchers of new visualization approaches should clearly define beneficial and unfavorable context aspects in order to support an application of their approach within a context-aware visualization framework. Thereby, the definition of context and its aspects is open and has to be extended in future research. Context aspects used in today's applications show only a small fraction of the potential, especially the availability of high-level context awareness and reasoning—like "user is currently driving a car and therefore must not be distracted with complex interfaces"—will offer new kinds of (mobile) applications.

In contrast, an effective utilization of available context data is sometimes difficult to find. Furthermore, results of an interpretation of multiple context characteristics may mutually exclude themselves, which motivates some sort of prioritization. Similarly, applied context-aware visualization techniques might depend on or require further context information in order to produce meaningful results; they even may trigger additional visualizations that have to be integrated. Therefore, future research has to revise the context evaluation and interpretation process in order to support sophisticated real-world scenarios for context-aware visualization applications.

In general, context-aware applications and devices are designed to support users and *automatically* sense the users' intentions. However, users might get confused by such context-controlled behavior, especially if context characteristics are interpreted differently than users expect. Thus, context-controlling aspects should be obvious to the users, i.e. behave as users expect, and for practical applications extensive user evaluations are necessary.

6 Application and Results

The proposed generic framework for context-aware visualization is designed to efficiently develop applications with context-aware presentation functionality. In this summarizing chapter results and evaluation of various prototypical example applications that make use of the framework are discussed. The considered scenarios range from applications supporting tasks in everyday life to specific tools in manufacturing environments, primarily utilizing mobile devices.

Further, issues that may arise when deploying context-aware systems in real-world scenarios are highlighted; these are of technical but also of legal and ethical nature. Thereby, privacy of context data among other things is an important factor; although the discussion concentrates on location information, similar problems are expected for other context aspects.

6.1 Context Awareness in Praxis

Deployment of sophisticated context-aware systems for mass-market—as currently discussed and foreseen in research [38, 163, 47]—has still not yet happened. Many projects show the advantage of context-awareness, but only few address the fact that utilized and required technology is often suboptimal. Thus, real-world applications—probably targeted for mass-market—demand more advanced and reliable basic technology. But not only technical problems have to be solved also legal and ethical restrictions of context awareness have to be clarified before such systems can be put into operation.

Definitely, many applications benefit from a context-aware behavior ranging from entertainment to information and finally to professional applications. This wide scope and the involved user classes motivate an interdisciplinary view on problems, possibilities, and implications that occur when context-aware systems are deployed.

6.1.1 Technical Problems

Context awareness is primarily based on the acquisition of diverse sensor information to capture data of physical and virtual entities. Utilized sensors play a major role within real-world scenarios as capabilities and limitations have a direct effect on the experienced usability of the entire system. Together with E. Westkämper and L. Jendoubi required technology—with focus on sensors—and limitations are identified, exemplarily with respect to location and identification

context data for applications in a smart factory [207].

A primary matter for applications in smart factories are object tracking and object identification. Current projects most commonly realize identification and tracking of objects via RFID technology. For the considered scenario—tracking of objects within a manufacturing environment—several restrictions are placed on the communication facilities: A small form factor is required in order to be able to attach tags even on small tools like milling cutters. Further, a reliable operation is essential even within manufacturing environments where influencing magnetic or electric fields caused by production machines are present.

In fact, RFID tags for identification are robust against errors, but for advanced applications the required accuracy of location information can hardly be achieved with this technology—RFID location information is only an approximated distance relative to the currently detected tag. Systems for global positioning that determine absolute 3D positions are based on computer vision [4], ultrasonic [101], or ultra-wideband [196] technology, but measurements showed that, e.g., ultra-wideband-based location systems offer an accuracy of only ± 50 cm in manufacturing environments vs. 15 cm as stated under optimal conditions [196]. Even if optimal conditions are met, applications like augmented reality require a much higher accuracy (location $< \pm 1$ mm, orientation $< \pm 1^\circ$) and a real-time update rate, in order to keep the virtual and physical information aligned. In addition, location tracking in smart factories has to support a huge number of objects that have to be monitored simultaneously, in [207] it is estimated that $\sim$2000 updates per second are required for currently active tools within one site. On top, less frequent updates are needed for tools in stock or at remote sites. Currently, these demands on accuracy and number of tracked objects cannot be solved by any of the aforementioned systems.

Adequate human computer interface technology is a further important aspect. Interfaces have to be designed to support huge amounts of data efficiently and adapt according to context aspects providing an interface adequate for the current situation. An ergonomic usage of software and hardware is crucial: Displays, especially of mobile devices, should have high brightness and contrast so that an operation on various lighting conditions is possible. Most important safety regularities have to be satisfied, e.g. using video-see-though AR devices for context-aware visualization in hazardous environments is problematic as in case of system failure users might get harmed. Therefore, setups using optical-see-through devices are preferred as system failures only influence the augmentation maintaining the real-world view operational.

Required software-based data management of highly heterogeneous data formats, as common at manufacturing sites, is already addressed by data-federation systems like Nexus. An integration of diverse existing systems (ERP, CRM, etc.), which are already capable to manage partial data aspects of modern factories, is favored to accomplish a smooth transition to a smart factory.

Already this specific study on the deployment of context-aware systems within manufacturing environments shows that many additional technical details have to be considered when deploying context-aware research prototypes to real-world scenarios.

6.1.2 Legal and Ethical Restrictions

Besides technological limitations, an application of context-aware technology is further restricted by legal and ethical matters. Security and privacy of personal data is already an intensively discussed issue; this shows that users are concerned about their personal information. In contrast, context-aware applications are designed to acquire and consolidate huge amounts of information about objects and, more important, persons. Systems which track locations of persons, i.e. know their location at any point in time, are already an intrusion into privacy of users. Due to this, context-aware systems have to support concepts to maintain privacy.

At the workplace the same or even stricter rules apply, as employees are protected by the law and unions. In collaboration with D. Lucke, E. Westkämper, and O. Siemoneit the same scenario as considered in the previous section with respect to technical aspects—real-world deployment of next generation context-aware manufacturing techniques—were examined considering privacy aspects [132]. Due to privacy reasons, locations of workers must not be tracked using an infrastructure-based server system to overcome the potential risk of misusing information against workers. In contrast, with the availability of statistical data about workers' movement trajectories the factory layout could be optimized. Thus, maintaining privacy prevents the usefulness of context-aware applications to some extent, which makes clear that a proper definition of privacy is important to balance privacy vs. functionality of context-aware systems.

Privacy is the right of a person to decide whom to give which level of access to information about oneself. Privacy is important to the humanity as it allows personal freedom and the ability to build a personal scheme of life. However, social traditions in different cultures influence what is considered private. Furthermore, the privacy of persons may be overridden, e.g., to protect rights of other parties. The right of personal privacy is therefore balanced with rights of others or social norms, making privacy hard to quantify and evaluate.

When mapping the right of privacy to (future) manufacturing scenarios, it becomes clear that employees have the right of privacy even though they signed a work contract but certainly not without restrictions. In [132] the example is mentioned that employers have the right to supervise the efficiency of their workers, but this does not include a continuous on-line monitoring of a worker's activities.

A technical solution to implement location systems that enable context-aware applications and still maintain privacy is detailed in Section 2.2.4 and similarly presented in [132], also A. Butz et al. indicate the importance of privacy and propose an unidirectional communication for protection [30]. The semi-automatic nature of our approach exclusively allows users to decide on their own when and where the location is evaluated. Furthermore, the proposed configuration does not require a communication of location information to a server infrastructure, if a text-location database was previously acquired. Even if text queries are sent to a server, it is possible to obscure the true location via false requests; this, however, is only a non-optimal fallback solution and only possible since the data amount is very small.

Only a small fraction of possible legal and ethical restrictions of context-aware systems are discussed. Even though, it clearly shows that already for limited location-aware applications an interdisciplinary evaluation considering technical, legal, and ethical restrictions is required, which is even more severe for universal context-aware systems.

6.2 Context Awareness for Everyday Tasks

Most people already use context-aware systems frequently—popular examples are navigation applications—although they do not characterize them as *context-aware*. A more familiar term is location-based services/applications (LBS), which can be considered as context-aware with the restriction that only location information is interpreted. Work done in the field of location-aware systems is primarily targeted to acquisition and effective usage of context data, thereby primarily considering location information of mobile clients or users. The popularity of the global positioning system (GPS) and its corresponding navigation applications stimulates the development and research of additional novel features. However, the GPS signal is limited in precision and also cannot reliably be received inside buildings; therefore, indoor scenarios require different technology. The active badge system is probably one of the first indoor location-aware systems proposed by R. Want et al. [201]. Visualization approaches for location-aware systems and its corresponding position-annotated data are mostly straightforward (some are discussed in Section 5.3); the most basic approach are pins that mark locations on a map, as used in Google Maps [81].

Strictly speaking, location awareness considers only position, no orientation information. However, as orientation is tightly coupled with location it seems natural that both aspects are considered concurrently, but most location systems cannot provide reliable orientation information. The commonly applied fallback to calculate an orientation as derivative of the position is often highly inaccurate and therefore inappropriate, especially in mobile user scenarios. Infrastructure based location systems (see Section 2.2.1) may be used to overcome this limitation and provide reliable location and orientation information, but such systems are spatially restricted due to infrastructure installations and are often expensive.

The proposed semi-automatic location system, based on optical character recognition, provides location and orientation information at high accuracy using a non-infrastructure based approach, detailed in Section 2.2.4, which enables new applications even for non-professional markets. In the following, prototypical implementations of simple add-ons to partially existing solutions show the potential of context awareness also for common, daily tasks. Specifically, an indoor navigation system [132] and a real-world text augmentation system [55] are shown that make use of high-precision location and orientation information, both offering functionality that is rarely available on comparable systems.

6.2.1 Indoor Navigation

Classical navigation software for outdoor environments is also possible for indoor scenarios. Users can select different destinations inside a building and calculate a shortest route, starting at their current location. However, most known systems require additional hardware installations, as proposed by R. Oppermann and M. Specht in [154], or can only provide locations of about 3-5 m accuracy, e.g. presented by Small et al. [187]. Due to missing or highly inaccurate orientation information, users of navigation applications are still required to find directions, based on the given location, which is for some scenarios—e.g. symmetric buildings or similar looking corridors—a non-trivial task.

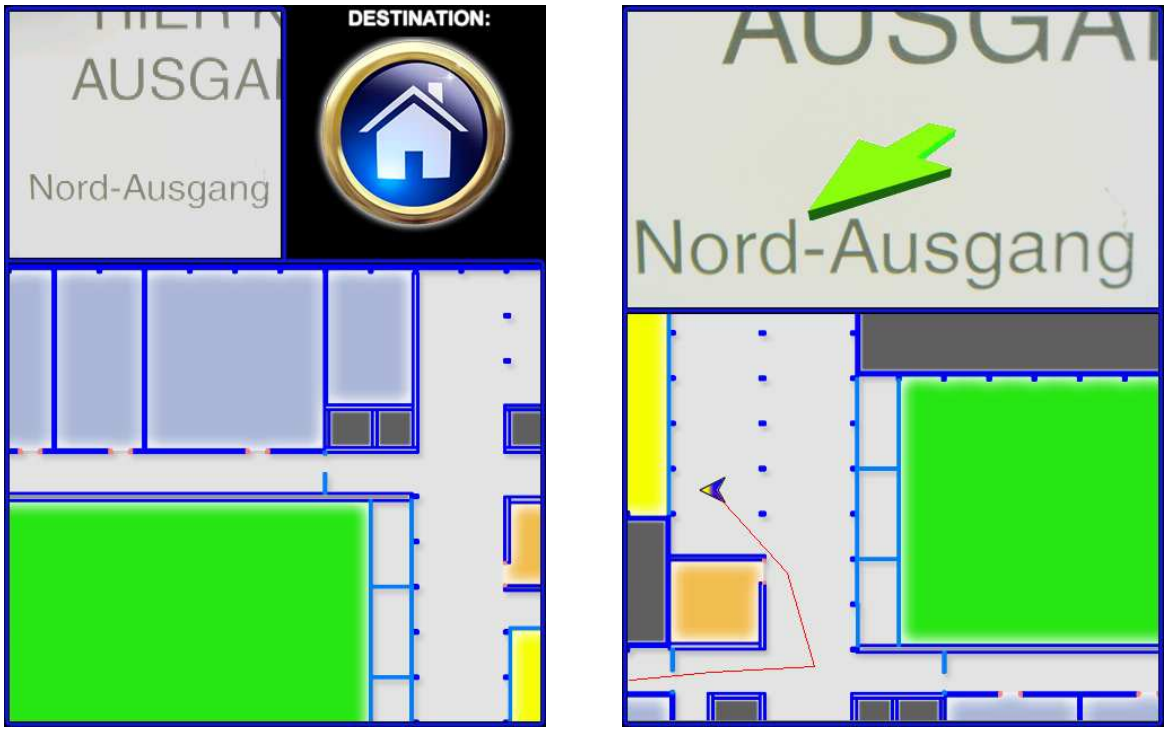
A. Butz et al. present a navigation system that shows different navigation hints considering available accuracy of the location and orientation information, cognitive restrictions of users, and technical restriction of the display device [31]. Their system is based on infrared beacons and achieves an accuracy of 20-180 degree for orientation information and probably 5 m for location, mainly dependent on the number of externally installed beacons and their emitting intensity. Their system specifically copes with inaccurate orientation information via different presentation styles. However, no fine-grained direction information can be given to users, which might work for office-style buildings with unique corridors and small rooms, but open areas, halls, or manufacturing sites cannot be supported conveniently. An extension is proposed by J. Baus et al. to adapt the graphical rendering of a navigation route, dependent on the observed location accuracy and the user's speed [14]. Additional location systems have been integrated, but benefits of accurate orientation information are not considered.

Figure 6.1a shows a screenshot of an example indoor navigation application. The lower part shows a floor map; the navigation destination is displayed in the upper left. As the application is tightly coupled with the proposed OCR-based location approach, a live camera image can be seen in the top left as preview for capturing text phrases. Note that in this state no location information is available, therefore no navigation hints can be given. Using the aforementioned techniques, only a roughly approximated position may be marked into the map to reflect the user's location.

In contrast, the proposed OCR-based location technique discussed previously (Section 2.2.4) can provide highly accurate position and direction information without requiring any additional hardware installations. This way, the user's view direction and location when the image was captured—usually just a moment ago—can be shown in map drawings, which help them finding their orientation relative to the map (Figure 6.1b, yellow-blue pointer, bottom part).

Utilizing an accurate orientation for visualization of navigation information also offers the possibility to introduce real-world aligned augmented reality techniques, thereby eliminating the requirement of users having to figure out directions themselves. Augmenting the image, which was captured with the client, requires a consistent visualization of the navigation symbols, relative to the orientation of the text entity seen in the image. The navigation direction is therefore projected to a plane whose orientation matches the physical text entity's pose. The geometric information about the plane orientation is just additional context information, determined during the location determination process. Afterwards, the navigation hint is rendered accordingly as an overlay onto the captured real-world camera image as shown in Figure 6.1b, upper part.

With such a technique, users only have to face a piece of text and capture it to get navigation hints aligned to their current location and viewing direction. However, a requirement is that the captured text phrases are unique for the area served by the system. To overcome this limitation, feedback to the user can be given—e.g. display hints of possibly unique texts in this building or directional hints where partially equal texts differ—or a hybrid approach can be applied using, e.g., few infrared beacons to separate sections of a building. Users may be informed of multiple possible locations of a captured text showing the uncertainly of each based on the string matching error and former positions, and the time period between, or statistics on likely user locations. Already depicting the density distribution where the captured text occurs most often may help users to conclude their location.



a) b)

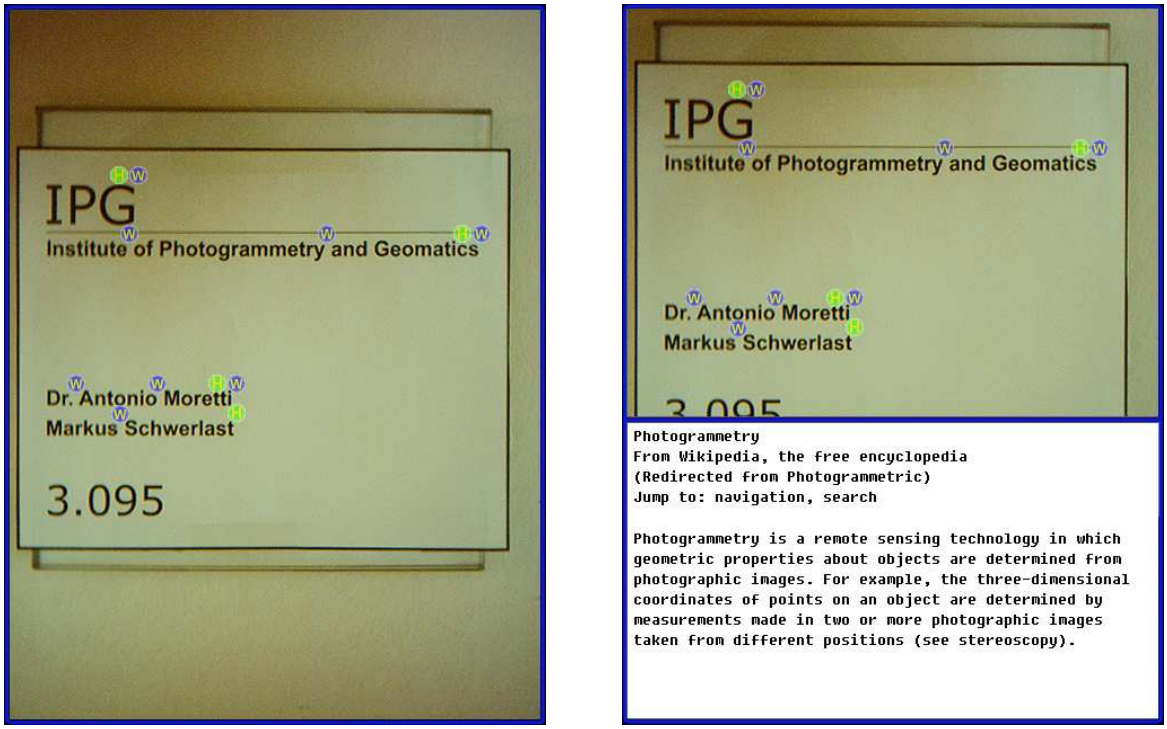
Figure 6.1: Navigation client interface. Illustration a) shows a real-time camera view with destination and floor map. An augmented view after the image capturing and location determination process shows the direction relative to the captured image—upper part of b)—and a map with location, orientation, and route information in the lower part of b).

6.2.2 Information Augmentation of Real-World Texts

In addition to the commonly offered navigation support of location-based systems, also other applications are promising. Utilizing characteristics of the proposed OCR-based location system, augmented reality methods can be used to provide an intuitive interface for additional information about real-world text phrases. In a similar way as the systems proposed by M. Koga et al. [118] and L. Jagannathan and C.V. Jawahar [102] for translation of texts to other languages, the text recognition process may attach additional metadata to the recognized text. Even further, dynamic information can be generated, based on multiple context aspects, e.g. client device type, user settings, time, date, etc. Using small information icons that are displayed using AR methods an intuitive user interface can be implemented where users simply have to click on the icons that are embedded in the captured image to access the additional data.

The required information and image-space positions for overlay icons are automatically generated by the text recognition process and are attached to the location information as additional context information. As mentioned in Section 2.2.4, the text-string database allows defining metadata related to text entities. The implementation of the prototype makes use of this feature to support hyperlinks to websites within printed real-world texts (H icon) that are only displayed

if the context information reports that the device currently has an internet connection available. In addition, translations are generated to a user-defined target language on a simple per-word basis (T icon) and a search query can be initiated on the free web-based Wikipedia[1] encyclopedia (W icon), again W icons are only shown if an internet connection is available. The left image of Figure 6.2 illustrates the information augmentation for real-world texts; after clicking on the W icon, right above the word *Photogrammetry*, the application view changes to show the description returned from Wikipedia (Figure 6.2, right, only the first paragraph is shown). Other icons are handled similarly; e.g. by clicking the H icon the associated web site is shown in the lower part of the screen. Note that the translation functionality is deactivated in the screenshots, otherwise each word would have an additional T icon associated.



a) b)

Figure 6.2: Augmented view of a captured text image. The icons (H and W) mark clickable interaction widgets to display further information (shown in b) for *Photogrammetry*). H refers to an internet homepage and W shows the first paragraph of the corresponding entry in the Wikipedia encyclopedia.

The demonstrated interplay of captured images and direct interaction on augmented texts characterizes an intuitive user interface to access relevant information of the user's current environment. The interaction is thereby reduced to single clicks/taps in combination with manually capturing images of text phrases. An accurate location and orientation information of the mobile device, relative to the text entity, is crucial in order to precisely align the augmentation icons

[1] Wikipedia is a multilingual, web-based, free-content encyclopedia project: http://www.wikipedia.org

within the text, which again highlights the need for accurate location and orientation information for novel location-aware applications.

6.2.3 Findings from Prototypes

The client prototype is implemented using a Windows Mobile 2003 PDA with an Intel XScale PXA263 400 MHz CPU, programmed using C++. The application uses direct screen access for drawing into the 480x640 resolution framebuffer and, thereby, achieves good interactive frame rates of about 6-12 frames per second (e.g. when scrolling the floor plan). The prototype was demonstrated during a public event at the university where visitors were allowed to try the system and navigate themselves inside the building. Most of the participants found the system very useful and were able to operate it instantly; they also liked the idea that no additional hardware is required and the client application could even run on most modern camera-equipped mobile phones.

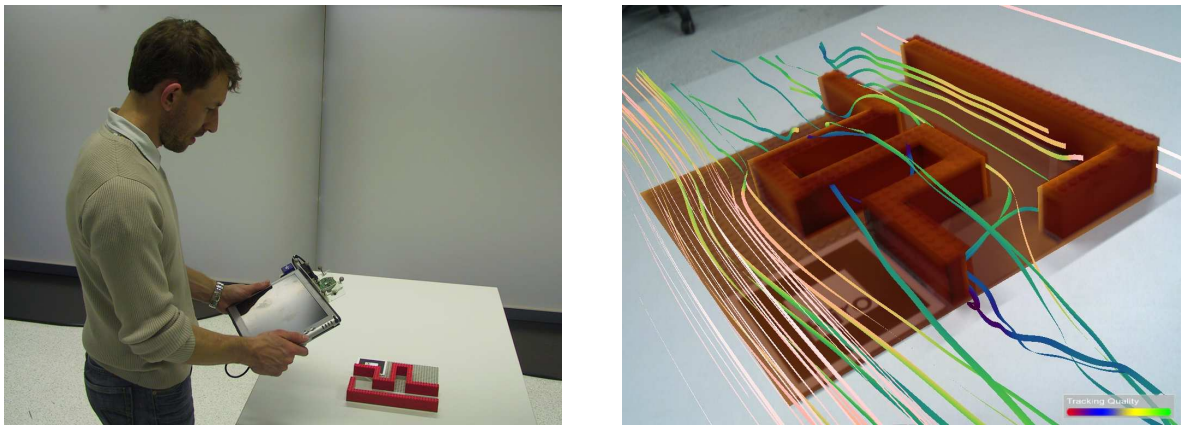
The proposed intuitive mobile applications support users in accessing information and navigating inside buildings, which demonstrates the potential of location-aware systems. Furthermore, the advantage of widening the definition of *location awareness* with additional context aspects is exemplified by just adding highly accurate orientation information, thereby already new applications are possible. This motivates further extensions using other context information to develop sophisticated context-aware applications and corresponding visualizations.

6.3 Context-Controlled Scientific Data Visualization

Visualization is and will be increasingly important as measurement techniques and simulation methods improve and provide enormous datasets, possibly including multiple time steps. The diversity of existing visualization methods is overwhelming often overstraining users when they have to decide which approach will provide best visualization results.

As already motivated in Chapter 4, the proposed framework and corresponding context-aware visualization techniques are able to support users in this task by evaluating available context information to control the selection and adjustment of visualization techniques. For evaluation, a practical example considering the task of vector field visualization—as an example of scientific data visualization—was build. A mobile tablet PC is used to implement a so-called AR window as shown in Figure 6.3.

The application is operated interactively by users, as depicted in Figure 6.3a, to explore aspects of the vector field. Note that the vector field represents a simulated air flow through the shown Lego scenario. The key concept of the prototype is to minimize the need for user interaction and instead make use of available context information to control the application including utilized visualization techniques. Typical for AR-like systems, location and orientation are considered, but additionally the distance of the viewer to a point of interest (POI)—the prototype defines the POI to be the center of the flow field—and the quality of context data are evaluated. Utilizing these additional context aspects allows the application to be operated naturally,



a) b)

Figure 6.3: Flow visualization using an augmented reality prototype. A mobile tablet PC is used as a window, as shown in a), to examine the flow behavior of the scenario. The device displays a life video image of an attached camera augmented with a flow visualization technique b).

i.e. only via physical adjustment of the mobile device, thereby visualization techniques automatically change or adjust accordingly to achieve the most pleasant visualization result.

6.3.1 Quality of Context Information

The quality of data—its correctness, reliability, or consistency—is a very important context aspect. Not surprisingly, quality of context is an active field of research in the Nexus project [38]. A generic concept to quantize quality of arbitrary context information, namely a generic metric, is not yet known. Therefore, specific context aspects are examined for which a definition of quality is well understood, e.g. for location information an error range $\pm$2m could be used to describe the accuracy of information.

In the presented prototypical example data qualities of utilized tracking systems are interpreted and used to control the flow visualization. Two tracking systems are used: the A.R.T. tracking system [4] and the AR-ToolKit [110]. For evaluation, a normalized tracking error $[0..1]$ to represent no/incorrect pose up to highly accurate pose estimation that is reported by location systems would be preferred, but none of the utilized ones offers this functionality. Even more severe, the current version of the A.R.T. tracking software provides only a Boolean error value: *true* if a marker is detected, *false* if not. This shows that even for such a limited example the quality cannot easily be evaluated. A theoretical, generic solution to estimate and handle errors of pose estimations is presented by E.M. Coelho et al. in [37]; however, the problem of retrieving any error measurements from an actual tracking system is not addressed. Therefore, a simple heuristic to evaluate a normalized tracking quality in realtime is used instead, as presented together with M. Kreiser in [57].

The pose estimation and its quality is evaluated for each frame; all tracking systems that provide valid pose estimation at the currently calculated frame are considered. A difference in orientation and location between the utilized systems is calculated and stored so that the sys-

tems can be synchronized to each other. This way, a tracking system that is known to provide highly reliable and accurate data can dynamically calibrate less accurate systems, whenever both positioning systems provide a valid estimate.

A normalized value reflecting the currently observed quality of a pose estimation reported by the AR-ToolKit is calculated based on: the confidence value, which describes the probability that the marker detection is correct, the size of the marker in the captured camera image used for detection, and a timer to prevent erroneously detected markers in isolated frames. In contrast, the A.R.T. system does not report any usable confidence or quality value; therefore, a constant quality is assumed but below the accuracy of the AR-ToolKit. Due to the aforementioned synchronization, the quality is increased for a time period after a synchronization event with the more accurate AR-ToolKit and lowered again afterwards. The normalized per-tracking system quality values are afterwards used to fuse the individually reported pose estimations and are additionally combined to a single quality value (a more detailed description is given in [57]).

6.3.2 Context-Quality Aware Flow Visualization

Multiple context-aware vector visualization techniques are used to realize the prototype, namely streamlines, stream ribbons, color-coded surfaces, and line integral convolution on surfaces (detailed in Section 5.3.3). Each technique adjusts itself to the viewer's position and orientation, i.e. according to the tracked table PC. Thereby, an accurate alignment of the visualization in relation to the physical scenario is crucial in order to achieve a pleasant mixed-reality experience for users. Although available tracking systems have notably improved, still their accuracy is limited. Instead of improving the tracking performance to achieve higher accuracy, the problem of occurring misalignments of visualizations vs. the physical scenario is tackled by applying different visualization methods that are more or less *alignment tolerant*.

In Section 5.4.1 alignment tolerant visualization approaches are introduced using the example of streamlines/-ribbons that are selected to minimize occlusion artifacts due to misalignment of occluder geometry. The prototype makes use of this functionality to fine-tune the visualization for moderate changes in the observed tracking quality, i.e. if the quality decreases then only streamlines with increasing distance to the occluder are shown. More drastic changes in the quality enable entirely different techniques; other than streamlines or stream ribbons, i.e. an accurate pose estimation allows using direct visualization overlays on physical surfaces. As mentioned, a further controlling aspect—primarily introduced to minimize the need of user interaction via an interface—is the distance of the viewer to a POI.

For engineers, different aspects of flow data might be important: an overall impression of the global flow behavior, a detailed view of specific parts of the field, or a dense representation of the flow-surface interaction. Visualization systems that offer techniques to emphasize these aspects exist but have to be explicitly controlled by users, to achieve the desired result. In contrast, the presented prototype makes use of the viewers distance to control the visualization: If the examined flow field is relatively far away from the viewer, it could be assumed that the user wants to get an overview and is not interested in fine details, therefore streamlines are used. At medium distance, more information is conveyed by replacing streamlines with stream ribbons and introducing color-coded surfaces to show the velocity magnitude on physical surfaces. For

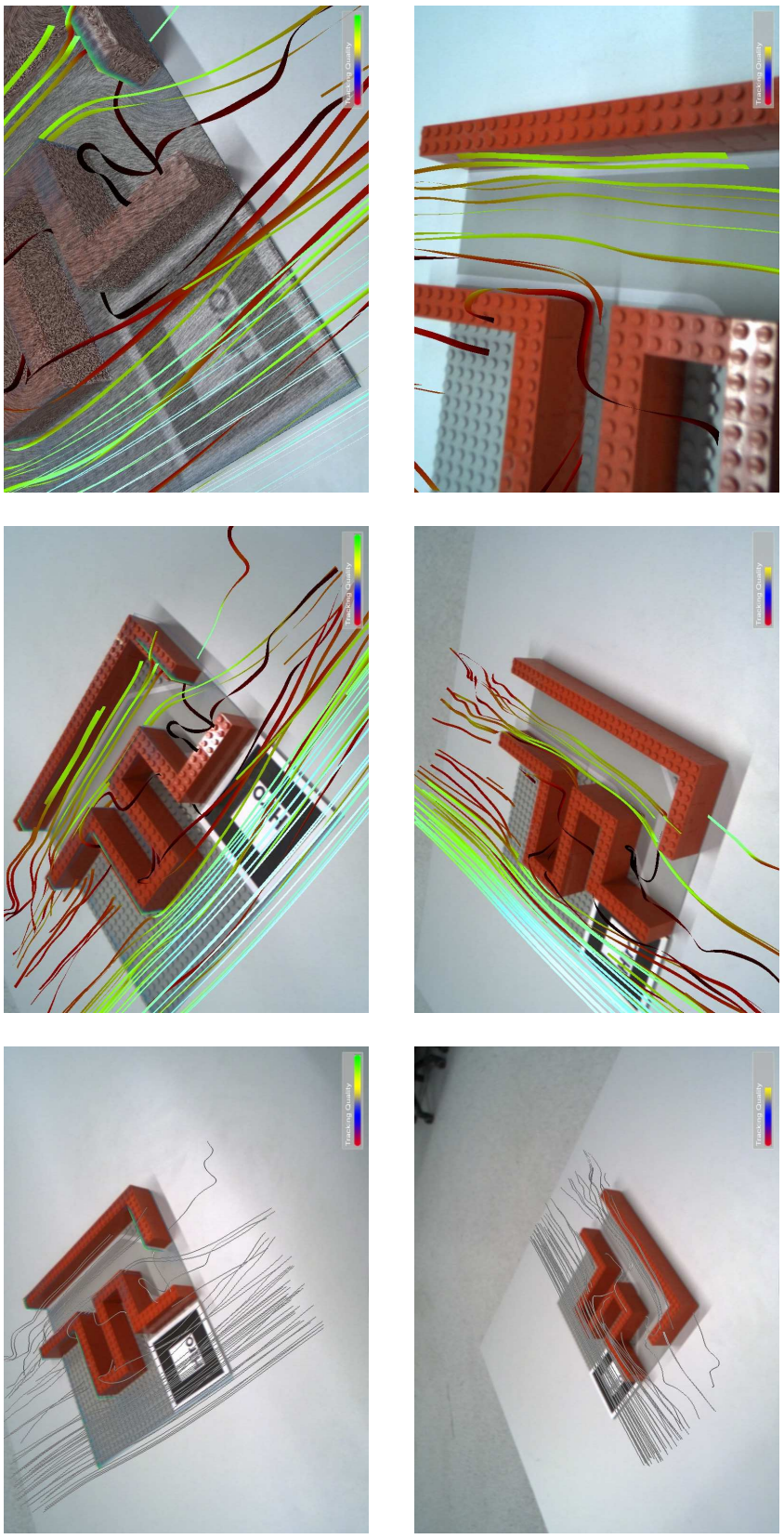


Figure 6.4: Snapshots of visualizations for an approaching viewer (from left to right). The upper row of illustrations show renderings if highly accurate pose estimations are available; the bottom row depicts visualizations for non-accurate tracking conditions.

close-ups, additional fine details are shown using the LIC approach mentioned in Section 5.3.3. As not all approaches are equally tolerant to alignment errors, the observed quality governs the selection process and ensures that only visualization techniques are utilized that are adequate for the currently available pose estimation accuracy. Figure 6.4 shows two image sequences of a viewer that approaches the POI. The top sequence depicts the resulting visualization if the pose quality is high; the bottom sequence shows results for low tracking accuracy.

Using the framework for context-aware visualization such context-aware behavior, as presented in the prototype, can simply be achieved by implementing the appropriateness weighting function accordingly, e.g. by using simple thresholds on the tracking quality context to return appropriate/not appropriate. Context-aware fine-grained adjustment of each technique is handled by each visualization template's context-update mechanism, also detailed in Section 4.2.3. As the framework allows an integration of custom context-switching nodes, even multiple visualization techniques presenting the same data—e.g. streamlines and surface LIC—can be used concurrently; the context-switch simply activates the two most appropriate visualizations for a specific data entity.

6.3.3 Usability of Context-Aware Scientific Data Visualization

For augmented reality and, in general, for context-aware applications, highly interactive frame rates are required in order to maintain synchronization of physical and virtual information. The overall performance of the system depends on the currently activated visualization method(s) (GPU load) and the utilized tracking system(s) (CPU load). Various hardware configurations have been evaluated including laptops and desktop machines. Even on moderate hardware, a single core Intel Pentium 4 CPU running at 3.4 GHz equipped with a Nvidia GeForce 7800GTX GPU, the performance at an image resolution of 1024x768 pixels is still highly interactive. The most resource-demanding configuration—line integral convolution combined with stream ribbons—only took 26 ms per frame resulting in approximately 39 fps.

The high quality and good performance account for a great user experience when operating the prototype. During first preliminary tests, non-expert users were surprised by the simple, natural way of flow-field exploration. Interaction by orienting and moving the tablet PC freely helped users greatly to intuitively understand and make use of all the features provided by the system. The use of further context information—besides position and orientation—to control the visualization was accepted by the candidates and used to examine different aspects. However, the relation of available tracking quality and utilized visualization technique was not obvious to the users, which is an indication that misalignments are less noticeable when the proposed alignment-tolerant methods are used.

Context-aware visualization is beneficial for scientific visualization tasks, as verified on the example of flow visualization. Active research, especially in scientific data visualization, result in more and more techniques, each having advantages and disadvantages in specific situations. As a consequence, users hardly know all available options and therefore might apply a suboptimal visualization to given datasets. Context awareness can alleviate this task by utilizing context information that describes the current situation and choosing visualization approaches accordingly. However, current prototypical context-aware applications are often limited to few context

aspects like augmented reality systems consider only location and orientation; in contrast, the proposed system shows advantages of considering further context information to enable richer functionality.

6.4 Scenarios in Manufacturing Environments

In order to demonstrate the advantage of context-aware visualization also for industrial applications, tools have been developed in close collaboration with the Institute of Industrial Manufacturing and Management[2] at the University of Stuttgart. Their research on smart factories envisions future manufacturing sites, based on a federated data source as information backbone. Therefore, the framework for context-aware visualization—proposed in this thesis—can easily make use of this data source to provide context-aware visualization for manufacturing applications.

Different fields of application are addressed ranging from tool maintenance, to site planning, to manufacturing condition monitoring. It is further shown that different client devices can be used—as naturally supported via the framework—to realize various use cases.

6.4.1 Tool Maintenance

In research presented together with E. Westkämper and L. Jendoubi in [207] applications of context-aware systems in manufacturing environments are discussed and a simple tool maintenance application to estimate tool conditions has been developed. Although only orientation is evaluated from the available context information, users already benefit from the added value. The prototype supports non-expert users in evaluating the condition of tools. Figure 6.5a shows a photograph of the prototype in use: A worker places a tool in the examination port of a tool box and can use the mobile prototype to examine the tool. On the display, annotations are shown that give hints to the user how to evaluate the tool's condition. The prototype implementation utilizes Nexus as an underlying context-aware framework which provides additional context information regarding the tool: previous usages, duration, alternative tools, etc. Thereby, the user's view to the real and virtual tool is automatically synchronized, simply by evaluation of inertial orientation information as discussed in Section 4.5.

Using the same principal of orientation awareness, a further prototype was developed as a showcase application for Xybernaut[3] presented in Figure 6.5b. In this scenario users were able to virtually explore physical objects (not seen in the image). The rendering of the virtual counterpart is annotated and can further be decomposed to see the inside of complex devices, e.g. in the prototype a physical power supply could be virtually explored.

These minimal use cases were shown to various users. Most experts, non-experts, even non-technical users gave a positive feedback that the mental mapping from the virtual to the physical object and vice versa is simplified by the use of context information to automatically synchronize the view of the virtual and physical object.

[2] Institute of Industrial Manufacturing and Management (IFF): http://www.iff.uni-stuttgart.de
[3] Xybernaut Corporation: http://www.xybernaut.com

a) b)

Figure 6.5: Prototypes of orientation-aware visualization applications. A tool maintenance application is shown in a), and an information-annotation application is presented in b).

6.4.2 Factory-Layout Planning

According to E. Westkämper and L. Jendoubi [206] there is an ongoing trend to higher flexibility and shorter planning horizons for manufacturing in future factories. In fact, manufacturing facilities have a long life, whereas applied production configurations change within months. This discrepancy has to be solved by factory-layout planners in order to maintain efficiency of a factory. Insufficient accurate planning leads to missing equipment, unpunctual deliveries, etc. resulting in reduced productivity. Therefore, planners use simulation tools to analyze layouts, e.g. virtual 3D models of the manufacturing site, but frequent changes during the lifetime of a production system makes it hard to maintain an up-to-date digital model.

Together with E. Westkämper, L. Jendoubi, and J. Niemann a conceptual solution to this fundamental problem of future factories is proposed. In [208] this discrepancy between reality and digital data is addressed by linking context information, which contains intermediate results of an off-site virtual factory-layout planning tool, to an AR system within the manufacturing environment. This way, real-world data—i.e. a real-time video stream—is combined with the virtual planning data and users are able to judge the newly proposed factory layout within the physical environment. Figure 6.6 shows a view of a manufacturing environment where the placement of an additional robot is discussed.

On-site operators utilizing such a context-aware visualization can give instant feedback to the planning office, discussing sources of non-optimal or even invalid planning layouts. Further, augmentation of the real-world view using context information provided by, e.g., context-aware frameworks can show additional important information to evaluate a proposed configuration. For instance, installed supply pipes, electric circuits, or even walking/driving passages of high utilization might as well be an important fact that has to be considered for layout planning.

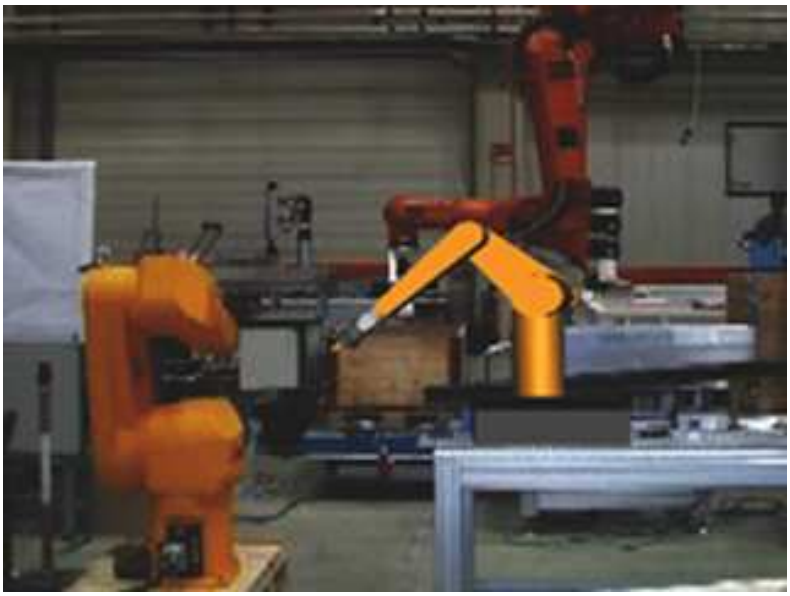
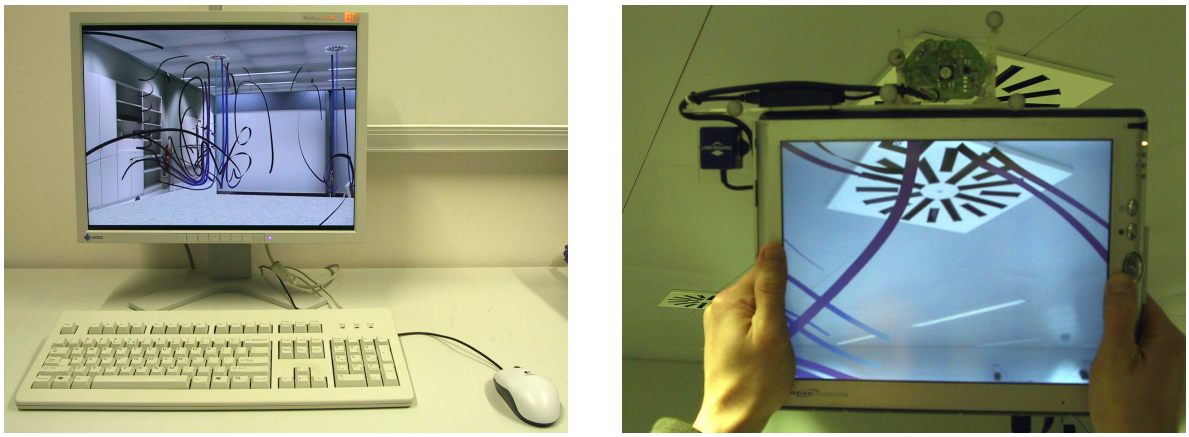


Figure 6.6: A prototypical example for digital shop-floor planning. An augmented real-world view of the factory shows the currently planned configuration, provided by actual context data from the planning office.

6.4.3 Planning and Evaluation of Manufacturing Conditions

Planning the layout of manufacturing sites is a non-trivial task, especially if sophisticated technology is utilized in the production environment. Similar to the scenario discussed in the previous paragraph, tasks like planning, evaluation, and monitoring of manufacturing conditions benefit from context-aware systems, especially from context-aware visualization. Complex production processes for, e.g., semiconductors require defined environment conditions during production in terms of air flow, particles per cubic meter, etc. Such specific conditions are maintained in so-called clean rooms using various ventilation systems and filters. In order to determine and evaluate the required configuration for custom installations, sophisticated flow simulations are applied that predict the flow patterns inside these environments [193]. However, for simulation an up-to-date detailed virtual 3D model of the clean room is essential to achieve accurate simulation results. Dynamic aspects, like moving humans or changes of the environment during the production life time, e.g. installation of additional storage containers, are often not considered and may therefore lead to undesired air flow characteristics inside clean rooms. To overcome this, visible fog can be blown into the room using foggers [7] to visualize the air-flow pattern. In contrast to simulation methods no virtual 3D model is required since the real environment is used, but this certainly requires a shutdown of production making it inappropriate for frequent evaluations.

With the availability of real-time context data and advanced context-aware visualization techniques mobile systems can be build to tackle the aforementioned problems. The smart factory, as defined by E. Westkämper and L. Jendoubi in [206], already manages all data relevant to the manufacturing via Nexus; therefore, the framework for context-aware visualization as detailed



a) b)

Figure 6.7: Context-aware data visualization for manufacturing environments. A visualization of air-flow data is shown using stream ribbons in a) together with a virtual 3D model to provide a spatial context of the data. A detail of the same data is depicted in b) using a mobile device with video-based real-world context.

in Chapter 4 can easily be used to build applications and visualizations to demonstrate the benefits, as proposed in cooperation with D. Weiskopf in [63]. Together with partners from the field of advanced manufacturing engineering a number of example visualization techniques were implemented, based on different basic concepts and deployed to different client hardware. The dominant context aspects that influence the visualization are data type, position, view direction, and client device capabilities. Visualization often benefits when related context information is also shown, therefore a combined visualization of scientific data and real-world video streams is supported. Naturally, both content types—real-world and virtual data—are adapted to the user's current position and direction to support AR setups. In addition, further production context data may be integrated, e.g. robot movement, material flow, workers entering/leaving per day, etc., to improve the utility of the system.

The examined example scenario is targeted for the design, evaluation, and maintenance of air ventilation systems, typically installed in laboratories or manufacturing environments. Direction and speed of the air flow or temperature are only a few important aspects that have to be considered, especially within areas where high-tech materials are processed. Simulation and visualization of these quantities are used to optimize the production, but evaluation and maintenance of the conditions requires expensive highly-qualified personal. In contrast, the proposed context-aware visualization prototypes can easily present the relevant data or information—adequate for different tasks—within the related spatial context, so that even non-experts can interpret the data to some extent.

Figure 6.7a shows an interactive visualization of the simulated air flow within the planned manufacturing environment. The simulation data is precomputed and stored within the AWM (augmented world model)—provided by the federated data source Nexus—and linked to its real-world location. The desktop application is based on the framework for context-aware visualiza-

tion, as detailed in Chapter 4, to access and visualize the simulation data in combination with other data, available in the AWM. The additionally presented virtual 3D model of the manufacturing environment helps users to understand the relation of the air flow to the environment. Additional data, e.g. temperature inside the room, can be displayed using volume rendering to investigate possible relations between temperature values and air flow behavior. Due to the utilization of graphics hardware, such a combined visualization still performs at interactive frame rates, even on moderate hardware. The total amount of data transferred to the desktop client is dominated by high-detail textures and adds up in total to roughly 12 MB for the visualization depicted in Figure 6.7a.

In the desktop application, real-world spatial context is artificially generated via a virtual 3D model of the manufacturing environment. Also interaction, i.e. change of viewing position and direction, is simulated via mouse interactions. Therefore, if the quality of the provided context data is not adequate, the resulting evaluation will not be accurate. Instead using smart mobile clients the system is able to replace the virtual environment with an image stream capturing the real-world conditions (see Figure 6.7b and Figure 6.8). Therefore, users are able to interpret and evaluate the simulation data within its relevant spatial context; misplaced interior, or obvious simulation errors are easily visible.

Basically, the very same techniques as applied for the desktop client can also be used to display the data on mobile devices. Due to the limited OpenGL functionality and performance of the utilized client device (TabletPC M1300, Intel 855GM 64 MB VRam) the system restricts itself to a limited number of stream ribbons to display the air flow or color-coded surfaces for the scalar temperature values in order to keep the visualization at interactive frame rates. The entire visualization is displayed in front of a live video stream that provides the real-world spatial context for the visualization. The resulting visualization, shown in Figure 6.7b, runs at 20 fps and requires an initial data transmission of roughly 4 MB to the mobile client for the entire scenario.

Furthermore, mobile clients are preferred for maintenance of conditions inside manufacturing environments as rearrangements or other changes of the environment can be captured. Even untrained users can easily benefit from such visualizations using their off-the-shelf smart mobile device like phones or PDAs. However, limited hardware of these devices often force to render the visualization on the server and use image streaming to transmit the final result to the client, which is automatically detected by the framework. For evaluation, a camera-equipped hand-held device (O^2 Xda Flame) is used to provide video-based real-world context (Figure 6.8) and to render the visualization as an overlay on top. However, due to restrictions, like small screen size, visualizations are adapted or even different for the same scenario. Furthermore, the knowledge of the operating users play an important role in the selection mechanism.

Benefits and Results of Context Awareness in Manufacturing

The detailed example highlights advantages and flexibility of context-aware visualization when supporting mobile users with different qualifications, different aims, and different hardware in inspecting and understanding huge amounts of possibly complex data. Visualization approaches developed according to the considered scenario effectively make use of the introduced framework and demonstrate the potential of interactive context-aware visualization. Developing aforemen-

Figure 6.8: A manufacturing-condition maintenance application running on a limited smart mobile device.

tioned applications is greatly simplified: Existing implementations of visualization techniques can easily be reused, extended, or modified to build rich tools for interpreting and analyzing heterogeneous context-based data, queried from a federated data source. Even if additional, not yet available techniques are required, the implementation effort is comparable to non-context-aware visualizations as specific context-aware functionality is mostly hidden by the framework. In addition, the consequent use of OpenSceneGraph to implement the proposed framework allows an easy adaptation of existing OSG-based visualization implementations to build context-aware visualization techniques. The scene graph concept further allows a combination of multiple different visualizations that show various data concurrently in a single image, although the individual visualizations techniques have been designed and developed independently of each other.

Overall, development time for applications that adjust their visualization in a context-aware fashion has been reduced greatly with the proposed framework. Techniques are developed once and are automatically available on various devices controlled by available context information. Most functionality for context awareness is covered by the framework so that, e.g., visualization in augmented reality applications are broken down in two context-aware visualizations: presentation of a live video stream—which is intrinsically based on the camera's location and orientation—and rendering of a desired data visualization using parameters of the physical camera.

An aspect not addressed by the illustrated prototype is the support of context-aware interaction, which is especially beneficial on mobile devices as shown in Section 4.5. However, with availability of sophisticated, future context-reasoning techniques—as foreseen in Nexus—higher-level context information will be available that describes situations of users and objects. This way, applications and visualizations can automatically change according the user's situation: E.g. user A explains a problem to user B who favors bar-charts vs. pie-charts therefore the visualization will make use of bar-charts to support user B in understanding the problem more easily.

6.5 Context-Aware Geo-Mashup Visualization

With the advent of online services like Google Earth [80] or Microsoft Virtual Earth [140] and their 2D counterparts, geo-mashup technology is freely available to ordinary internet users. Interestingly, their popularity increased rapidly in a way that both are now a de facto standard and often embedded in web pages to show the geographical relationship of some data. Both support the exploration of context-aware data: a virtual view defines a geographical section that is shown including annotations; most often simple pins are used. Furthermore, interactive navigation support allows a free adjustment the virtual view, the virtual camera on the data, resulting in interactive context-aware visualization.

Primarily, detailed surface height-field geometry, high-resolution orthoimagery, information pins, and for some regions 3D building models are state-of-the-art. Although such systems are *only* location-aware, they provide a well-grounded concept that is extended in this thesis to a context-aware data explorer. The developed prototype exposes the full potential of the proposed context-aware visualization framework detailed in Chapter 4. In contrast to aforementioned proprietary geo-mashup solutions, the prototype is based on an underlying federated data source enabling the possibility that multiple data providers can contribute to an aggregated data source. Thus, not only static information overlays, like a KML-file[4], can be integrated, but sophisticated dynamic visualizations are provided based on multiple different data sources.

As discussed already, the context-aware visualization framework is based on OSG and can therefore easily make use of OSG-based implementations and technology. For a location-aware data explorer the OSG terrain rendering functionality is utilized to implement tile-based level-of-detail terrain rendering. Texturing, geometry generation, etc. are handled automatically by OSG, even possible cracks of adjacent terrain tiles are removed. In combination with the framework's transparent support for mobile clients (see Section 4.4), a state-of-the-art (mobile) geo-mashup applications is realized. Figure 6.9 displays a screenshot of the prototype showing Stuttgart's city center including buildings. (The dataset is provided by the Institute for Photogrammetry (IFP) of the University Stuttgart.)

Using mid-range graphics hardware, like an Nvidia GeForce 8800GT, the rendering performance is approximately 27 fps for the shown visualization at a view port resolution of 1600x1200 pixels; for measurement an Intel Core2 Quad Q6600 CPU powered system was used, thereby the framework utilized multiple cores. If Stuttgart's entire city center with textured buildings is visualized then the frame rate is slightly reduced to approximately 17 fps. In contrast, for most mobile clients the geometrical detail of the scenario is too high; therefore, the framework automatically performs the rendering on the server and transmits an interactive image stream to the client, at the resolution that matches the client's display resolution.

Nexus, the utilized federated data source, considers not only location context but provides generic support for arbitrary context attributes. Thus, the proposed extension of geo-mashup visualization applications to a context-aware data explorer is also realized using the Nexus framework.

[4] Keyhole Markup Language: http://www.opengeospatial.org/standards/kml

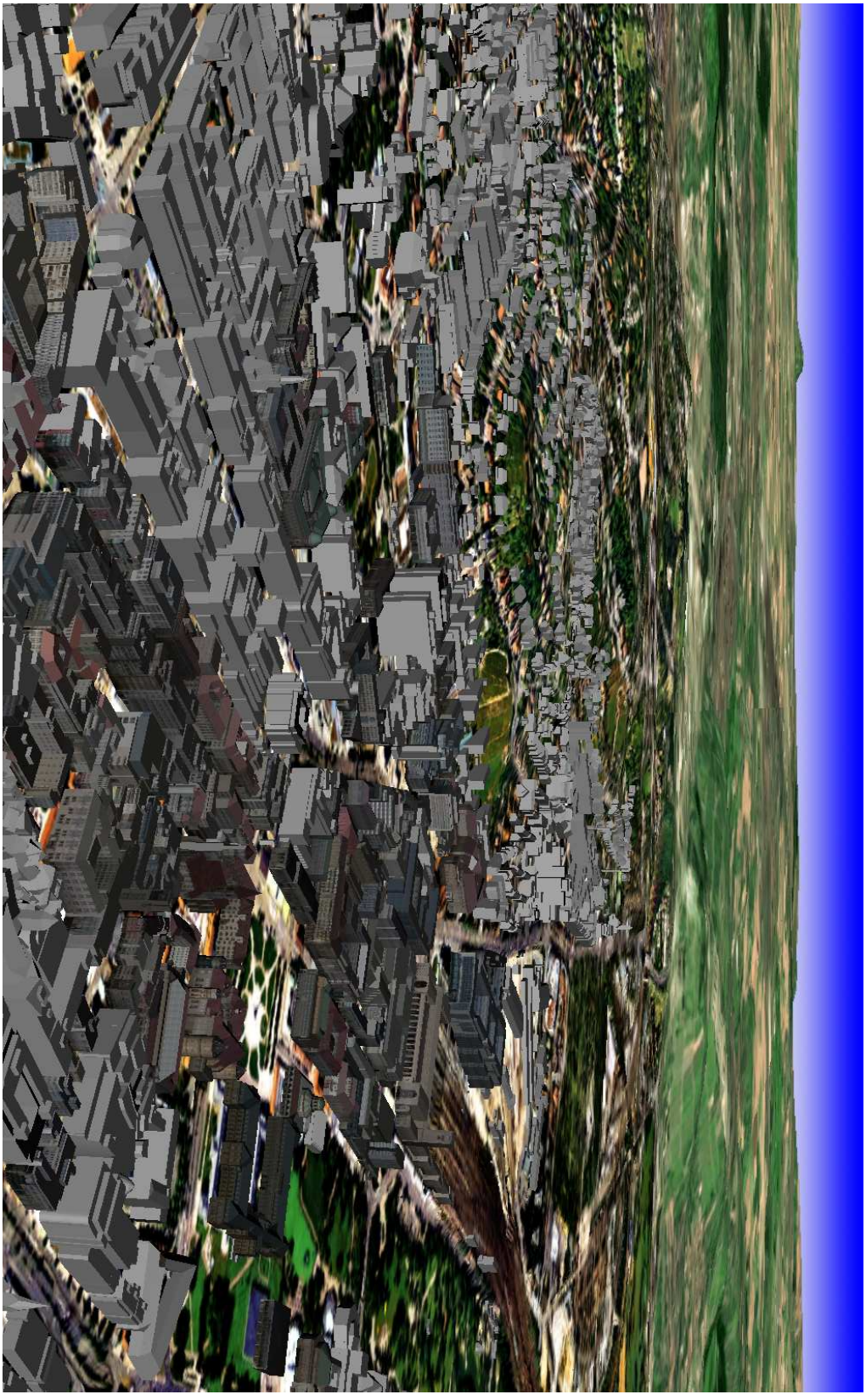
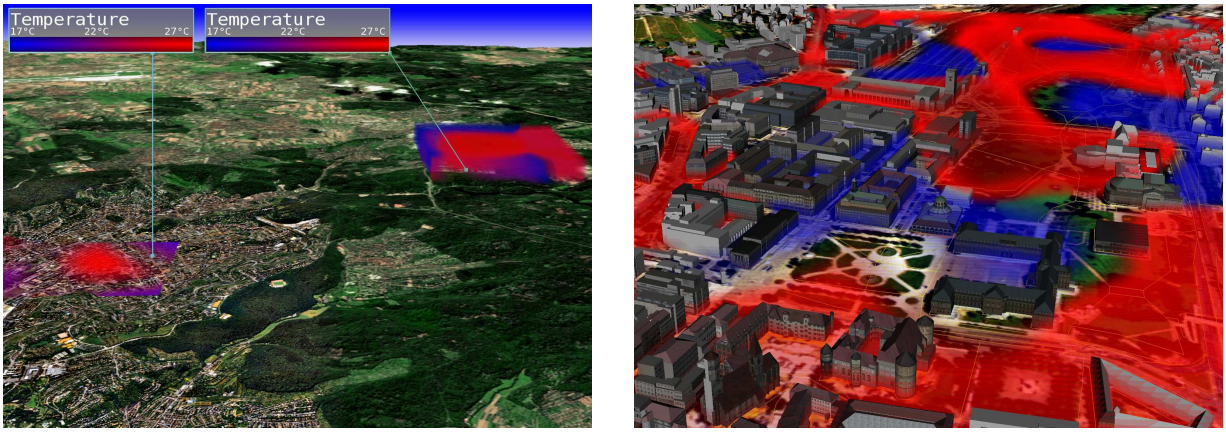


Figure 6.9: Screenshot of the geo-mashup prototype application showing terrain geometry, orthoimagery, and 3D buildings.

6.5.1 Geo-Mashup of Heterogeneous Data

No restrictions are placed on data type, data structure, or data amount; even visualizations composed of multiple, entirely different data entities are supported. Actually, the prototype supports a number of different visualization methods, in addition to 3D geometry including textures and GPU shader programs, text strings, scalar, and vector data types are handled. Thereby, all visualizations are implemented as templates (Section 4.2.3) and naturally available for other applications that make use of the framework.

The primary goal of the presented geo-mashup prototype is to offer visualization for *all* information provided by the underlying data source, namely the augmented world model of Nexus [145], automatically using an adequate visualization approach. However, extensibility is a key concept of Nexus and its AWM to allow an integration of arbitrary data types by various providers. This clearly eliminates the possibility to implement a visualization technique for each data type. The framework accounts for this problem with its transparent mechanism to incrementally cast data types to their parent types until a corresponding visualization template is found, in the last resort using a text box to display the data. However, if a real-world location is available, a geospatial reference pointer is shown that connects the information box with a geospatial position. Furthermore, most visualization techniques also instance information boxes to display additional information such as color tables or dataset identifiers.



a) b)

Figure 6.10: Advanced geo-mashup visualization showing different approaches to visualize scalar temperature data. The context-aware approach decided to utilize a 2D plane on the left part of a) and a 3D volume rendering visualization for the right part, as in this area 3D data is available. A combined rendering of volumetric temperature data with buildings and navigation data (yellow lines) is seen in b).

This way, the resulting geo-mashup of multiple heterogeneous data sources is more generic compared to the interface provided by, e.g., Google Earth or Virtual Earth [140] as representation and interpretation of more complex data types depends on corresponding visualization templates, which can be implemented without restrictions. Most important, the visualizations themselves are context-aware, not only considering a georeferenced position, but arbitrary context-aspects

like viewing direction, user preferences, client hardware, total amount of data, data entities within the current view, data format, and much more. Figure 6.10 presents examples for visualization of scalar temperature data; note that the context-aware behavior enables the application to present this data adequately because available context information is used to select and configure the visualization technique.

As an extension to Figure 6.9 the terrain and buildings are combined with temperature data. Figure 6.10a presents temperature data of two disjoined regions: in the left area only 2D data is available, thus the scalar values are mapped to the 2D earth surface using color-coding; on the right full 3D temperature data is available presented using volume rendering. The images are captured on a desktop client providing the required GPU and CPU support for volume rendering, otherwise the framework would have executed the visualization on the server or used a different visualization, e.g. a single 2D plane showing only an slice of the entire dataset. Figure 6.10b details a combined visualization presenting terrain, orthoimagery, buildings, navigation data (thin yellow lines on streets), and volumetric temperature data. Note how visualization techniques make use of info boxes to show additional information regarding the visualized dataset, e.g. the scale of the color-mapping.

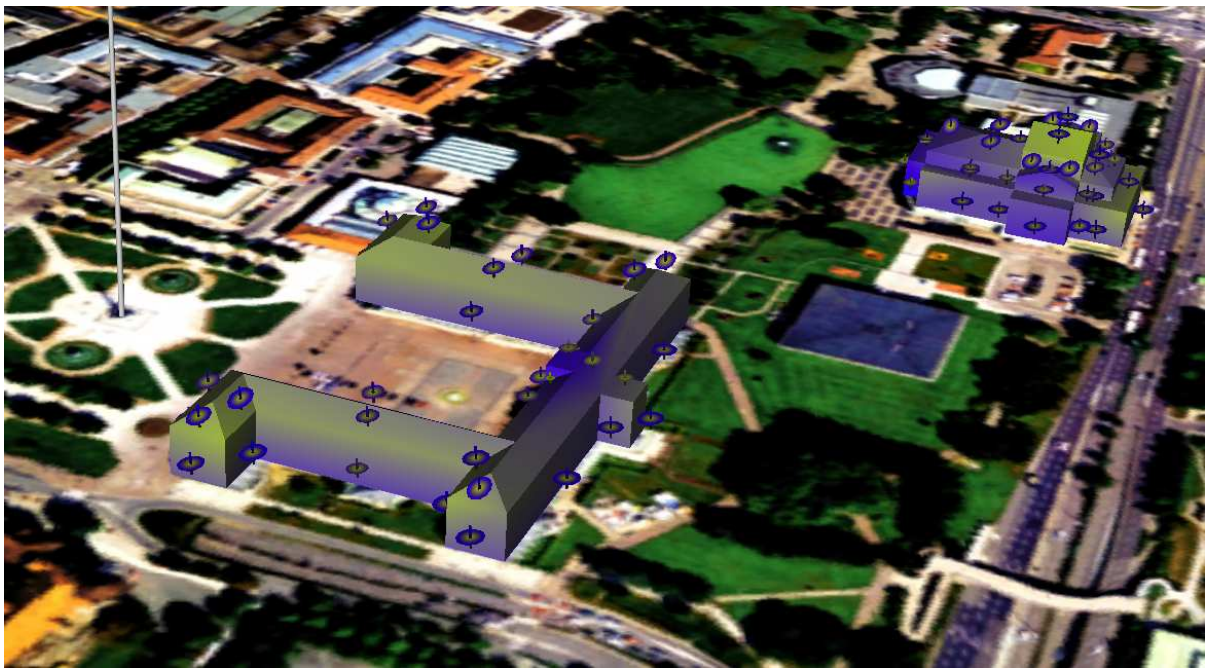


Figure 6.11: Orthoimagery of Stuttgart's city center combined with two buildings: the New Castle and Stuttgart National Theater. The consistency visualization shows the generalized 3D geometry models including glyphs that describe the consistency.

Entirely different, multivariate data can be visualized using glyphs, as introduced in Section 5.3.4. Glyphs are able to convey more information than just simple pins or arrows, as usually applied in Google Earth or Virtual Earth applications. The previously described consistency visualization for generalized building models can easily be utilized within the geo-mashup prototype

to present the data within its context making it easier to understand for users (see focus&context techniques in Section 5.1). Such a combined visualization is especially beneficial when, e.g., developing level-of-detail concepts where for each detail level the terrain representation has to match the generalized buildings, i.e. no gaps are allowed between terrain and buildings. Figure 6.11 shows a visualization of datasets provided by the Institute for Photogrammetry. Note that the alignment of buildings to the ground plan is done manually and can greatly be improved by automatic approaches as proposed by M. Peter in [160].

Again, the performance of the visualization is highly interactive (∼49 fps), even if additional buildings are shown in the same view approximately 31 fps are achieved, which allows users to explore the presented data interactively. Via navigation, the virtual camera can be manipulated and the visualization is displayed accordingly. Thereby, glyphs maintain their vertical orientation independent of the camera orientation to support users in recognition and interpretation of the glyphs. Considering the distance of the virtual camera to a building as part of the context allows introducing a level-of-detail concept where glyphs are merged together stepwise with an increasing distance (Section 5.4.2). Therefore, a compact representation of the consistency is achieved for faraway buildings, possibly only shown using a single glyph.

6.5.2 Generic Support for Mobile Devices

As the context-aware visualization framework used to build the geo-mashup application transparently supports mobile devices and automatically considers their properties, the presented visualizations are also available on smart mobile devices. Instead of adjusting virtual cameras in desktop applications, to simulate different location contexts, mobile devices are designed to be operated away from office. Thus, making them premium devices to experience location-aware or even more generic context-aware visualizations.

Figure 6.12: An advanced geo-mashup scenario where a 3D height field is combined with a flow visualization technique.

An example that shows the prototype's visualization capabilities on smart mobile devices is shown in Figure 6.12. A multi-resolution height field with overlaid navigation data and stream ribbons show the air flow inside a city; the visualization is rendered remotely and presented on a mobile device. The performance strongly dependents on the network connectivity and the visualization techniques selected by the framework, which tries to optimize for an interactive behavior, as discussed in Section 4.4. In contrast to, e.g., the 2D Google Maps application which is available on modern mobile devices, the proposed system offers more advanced interactive visualizations, optionally utilizing graphics hardware. Thereby, the visualized data is provided by a federated data source that combines multiple data providers to an open, distributed information store whereas related applications often are limited to a single, closed, proprietary database.

7 Conclusion

The amount of data—be it simulated, measured, or stored—will continuously increase in future and with it also different data structures and formats will arise. Using such information efficiently often implies various visualization techniques, capable of extracting and presenting relevant information contained in the data to users. Although, research in visualization will certainly provide brilliant solutions for new and existing datasets, still most of all techniques will have advantages and disadvantages for specific fields of application. Thus, users are forced to *manually* select—either based on expert knowledge or simply by trial and error—the most appropriate visualization technique for a given dataset.

In this thesis a context-aware approach is proposed to specifically address this fundamental problem of countless visualization methods for different data. The goal of this work is to present techniques that allow a (semi-)automatic selection and combination of arbitrary visualization techniques to adequately visualize data, dependent on the current situation (i.e. context aspects). Exemplary visualizations and prototype applications are shown to evaluate the effectiveness and benefits of context-aware visualization.

7.1 Contribution

Work in this thesis contributes to all relevant aspects for context-aware visualization techniques, especially techniques are considered that are a prerequisite for realizing context-aware methods. Thus, work starts in context acquisition and its interpretation and continues with its usage for selecting and configuring visualization techniques by discussing a novel framework. Visualization methods are realized that are selected and configured based on context information, which is evaluated within several prototype applications in different application domains.

When working with context data, often a framework to acquire, preprocess, and store context information is utilized—Nexus is used in this thesis. For specific context information, namely position and orientation, new approaches are presented: Different methods to interpret and utilize orientation data are shown to achieve simple context-aware behavior. Further, existing features, i.e. unique text strings, in indoor environments are utilized to build an infrastructure-less low-cost location system for mobile devices. Context-aware applications that are envisioned for mobile devices are often forced to get along with sensors that are embedded for context acquisition. Efficient approaches to combine multiple sensors and to capture interactive context are proposed,

that alleviate the limitation of non-automatic sensor adjustments and complete this thesis' work on context management.

In order to make use of context data for visualization purposes, a framework for context-aware visualization is developed. The focus on graphics hardware utilization enables the framework to efficiently process and visualize arbitrary data. Visualization techniques are integrated via a generic template-based mechanism that allows hardware-based and scene graph oriented implementations, even a transparent support for mobile devices is provided. As context awareness is especially relevant in mobile device scenarios, further research is performed in this thesis to optimize network transmission and preprocessing. To finalize the framework functionality, also a context-aware user interaction is discussed; based on previous work in orientation sensing and interpretation, a generic approach for orientation-aware user input is presented.

Focus&context is a related topic when researching context-aware visualization techniques, as they also make use of context information to improve the actual data presentation. Thus, focus&context is important for context-aware visualizations and supported by the introduced framework as it is capable to combine presentation of various data in a single image. However, work done in this thesis specifically addresses combined presentation of data and its relevant context, e.g. by zoomable user interfaces that allow to adjust the visible amount of context and focus information. A dominant aspect for context-aware visualization is the output device utilized to present the final visualization. Especially, for remote rendering setups where images are submitted via a network link and image size has a significant influence on the required bandwidth. An automatic adaptation is desirable but requires sophisticated technique for image adaptation in order to maintain high quality; some approaches are discussed within this thesis.

Context-aware behavior of visualizations is divided into two parts: context-aware selection among interchangeable techniques and context-aware configuration of selected approaches. Both aspects are addressed in the thesis and example visualizations are presented. Especially, a context-aware adaptation according to fast changing aspects is important to maintain synchronization of the visualization and captured physical context aspects.

The framework and corresponding visualization techniques are implemented and evaluated within several prototypes; navigation services, information systems, scientific data visualization, tools for manufacturing environments, and geo-mashup applications are considered in detail. Thereby, all of the examined prototypes provide unique functionality based on context-aware visualization, which motivates future research in this field.

This thesis introduces various concepts for realizing context-aware visualization with the goal to motivate the development of such techniques, which will result in a much wider acceptance of new visualization approaches, as users will automatically make use of them and will experience their benefits. The following brief workflow—depicted in Figure 7.1—may be used as a guide when developing context-aware visualizations. Researchers and developers may follow this workflow as a basic approach; however, due to the highly complex nature of context-aware visualizations, with manifold data sources and context aspects, a detailed plan, covering all eventualities, can hardly be found.

Dependent on the to-be-visualized raw data, available (non-context-aware) visualization techniques have to be selected; if no matching approach is available then a development of a new visualization technique is required. Development of novel visualization methods is a widely

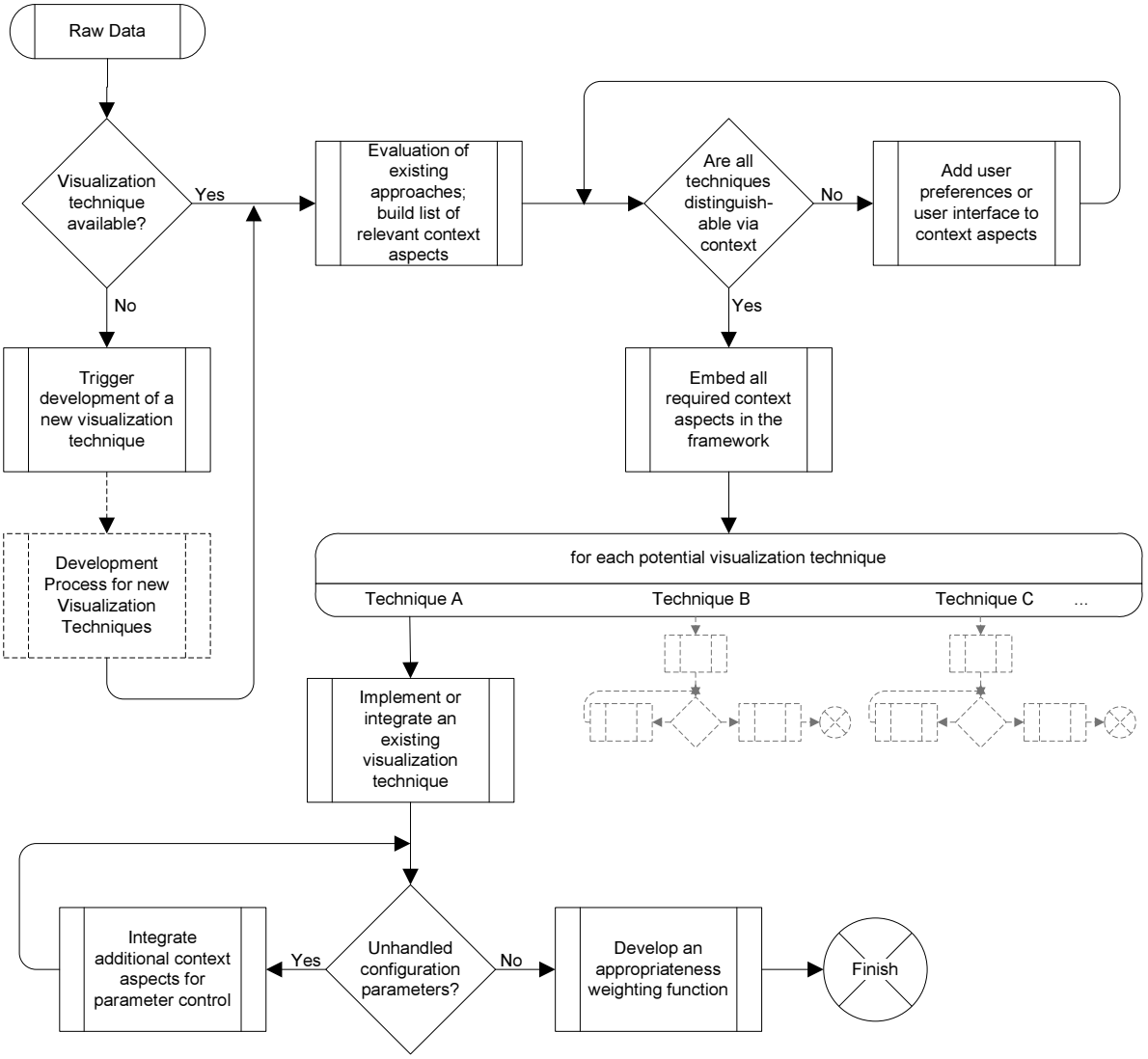


Figure 7.1: Decision chart for developing context-aware visualization techniques.

researched topic, which is not focused in this thesis; the reader is referred to [86]. However, having one or more appropriate techniques available, each has to be examined about information, aspects, or conditions that are beneficial for the corresponding technique; e.g. if only few data values have to be visualized scatter plots are adequate. All information, aspects, or conditions that are relevant to one or more visualization approaches have to be included in the available context information, considered by the framework. Still, techniques might exist that do not have advantages/disadvantages versus each other which are expressible via context or specific context aspects. In this case, the context is enriched via user preferences or user interaction elements, so that users are able to select their preference.

When all considered visualization approaches are distinguishable via context, each method can independently be included and used by the framework. Note that Figure 7.1 details this pro-

cess only for a single visualization, but the process is the same for each considered technique. First, the visualization algorithm has to be implemented in a basic non-context-aware approach. Almost all methods will have parameters that have to be adjusted, based on various criteria like data density, data rage, output intensity, etc.; context-aware visualization implies that these parameters are also automatically configured based on context information. Thus, a handler to adjust visualization parameters according to received context updates is also required (see Section 4.3). These context-aware parameter adjustments might require additional context aspects that also have to be included in the framework. Finally, an appropriateness weighting function has to be designed (see Section 4.2.3) and implemented based on all context information relevant for the visualization approach. Afterwards, the development of a new visualization template is finished and ready to be added to the framework for context-aware visualization.

7.2 Outlook and Future Challenges

Sensors, simulations, and databases continuously improve to capture more data, address larger problems, or manage bigger datasets. Thus, more and more information becomes available that needs to be processed in order to benefit from it. Visualization is the key technology to transform huge amounts of data in a way that the human visual system is able to understand it. This is further supported by using context information to adapt the visualization according to the considered problem, the current situation, or environmental conditions, as proposed in this thesis.

Future challenges will be both: development of adequate visualization techniques and approaches to automatically select and adapt them in order to achieve optimal presentations results for users. Techniques to efficiently handle huge amounts of data will be one focus; this will partially be addressed with upcoming faster hardware but more important by parallelization of algorithms. Future computing hardware will have increased parallelism; this trend is not only observable for GPU architectures but also for CPUs, as AMD's Opteron 6100 already integrated 12 cores [2] and AMD's upcoming Bulldozer CPU design will even feature 16 cores. Even more important Nvidia's current Fermi GPU [149] is based on 512 CUDA cores, designed for intensive, parallel tasks. When used as a graphics cards, its computation power is utilized to execute the rendering pipeline (see Section 3.1). However, the generic CUDA cores of Fermi are also able to perform arbitrary computation, providing a massive parallel CPU with 512 cores and a computing power of up to 630 GFLOPS in dual precision. Consequently, programming languages and APIs evolve that focus on parallel computation, Nvidia's CUDA [148], DirectX compute shaders [139], or the platform independent OpenCL [112] are the most established ones. Although these APIs are targeted for general purpose computations and abstract from the underlying hardware—be it a GPU or a multi-core CPU—still an in-depth knowledge of the utilized hardware is beneficial in order to design highly efficient implementations.

In contrast, context-aware systems are just evolving, but the benefits are promising: Computers will get more *intelligent* by sensing user's desires via context acquisition and react thereupon accordingly. In future research more context aspects will be considered, resulting in more sophisticated scenarios like, e.g., presentations that adapt the presented visual complexity depending on the participants' knowledge or user interfaces which simplify/restrict themselves if the operator

is driving. Visual perception is the dominant information source for humans, thus the goal of visualization is to prepare raw data so that the visual system is used in the most effective way to understand it, and context awareness has the potential to support visualization to achieve this goal. This thesis therefore inspires the visualization research community to design context-aware visualization methods considering and discussing advantageous and disadvantageous contexts.

Bibliography

[1] G. Abowd, C. Atkeson, A. Bobick, I. Essa, B. MacIntyre, B. Mynatt, T. Starner, et al. The Aware Home Research Initiative. http://awarehome.imtc.gatech.edu, last access: 13-Sep-2010. 20

[2] Advanced Micro Devices Inc. AMD. AMD Opteron 6000 Series Platform. http://www.amd.com/us/products/server, last access: 13-Sep-2010. 178

[3] Advanced Micro Devices Inc. AMD. ATI Stream Technology: GPU Technology for Accelerated Computing. http://ati.amd.com/technology/streamcomputing, last access: 13-Sep-2010. 59

[4] Advanced Realtime Tracking GmbH. A.R.T. http://www.ar-tracking.de/, last access: 13-Sep-2010. 32, 152, 159

[5] H. Aly and E. Dubois. Image up-sampling using total-variation regularization with a new observation model. *IEEE Transactions on Image Processing*, 14(10):1647–1659, 2005. 115

[6] Apple Inc. Apple iPhone. http://www.apple.com/iphone, last access: 13-Sep-2010. 33, 37, 88

[7] Applied Physics Inc. UltraPure Cleanroom Foggers, Model 2001/2010. http://www.appliedphysicsusa.com/clean_room_foggers.html, last access: 13-Sep-2010. 165

[8] S. Bachthaler and D. Weiskopf. Continuous scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1428–1435, 2008. 131, 135

[9] A. Baer, C. Tietjen, R. Bade, and B. Preim. Hardware-accelerated stippling of surfaces derived from medical volume data. In *EuroVis '07: Eurographics - IEEE VGTC Symposium on Visualization*, pages 235–242. Eurographics Association, 2007. 122

[10] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007. 20, 24

[11] J. E. Bardram. The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. In *Proceedings of the 3rd International Conference on Pervasive Computing*, pages 98–115, 2005. 24, 70

[12] L. Barkhuus and A. Dey. Is Context-Aware Computing Taking Control away from the User? Three Levels of Interactivity Examined. In *UbiComp '03: Proceedings of the 5th International Conference on Ubiquitous Computing*, pages 149–156. Springer, 2003. 37

[13] J. F. Bartlett. Rock 'n' scroll is here to stay. *IEEE Computer Graphics and Application*, 20(3):40–45, 2000. 34, 88, 89

[14] J. Baus, A. Krüger, and W. Wahlster. A resource-adaptive mobile navigation system. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 15–22. ACM, 2002. 155

[15] C. Becker and D. Nicklas. Where do spatial context-models end and where do ontologies start? A proposal of a combined approach. In *Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 48–53. University of Southhampton, 2004. 19

[16] B. B. Bederson, J. Grosjean, and J. Meyer. Toolkit design for interactive structured graphics. *IEEE Transactions on Software Engineering*, 30(8):535–546, 2004. 104

[17] B. B. Bederson, J. Meyer, and L. Good. Jazz: An extensible zoomable user interface graphics toolkit in Java. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 171–180. ACM, 2000. 104

[18] B. B. Bederson, L. Stead, and J. D. Hollan. Pad++: Advances in multiscale interfaces. In *CHI '94: Conference companion on Human factors in computing systems*, pages 315–316. ACM, 1994. 104

[19] D. Beier, R. Billert, B. Brüderlin, D. Stichling, and B. Kleinjohann. Marker-less vision based tracking for mobile augmented reality. In *ISMAR '03: 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 258–259, 2003. 50

[20] B. Bell, T. Höllerer, and S. Feiner. An annotated situation-awareness aid for augmented reality. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 213–216. ACM, 2002. 102

[21] M. Bernreuther and H.-J. Bungartz. First Experiences with Group Projects in CSE Education. *Computing in Science and Engineering*, 8(4):16–25, 2006. 71

[22] K. J. Blom. vjVTK: a toolkit for interactive visualization in Virtual Reality. In *EGVE '06: Proceedings of Eurographics Symposium on Virtual Environments (Short Paper)*, pages 17–19, 2006. 71

[23] H. Borotschnig, L. Paletta, M. Prantl, and A. Pinz. A comparison of probabilistic, possibilistic and evidence theoretic fusion schemes for active object recognition. In *Computing*, volume 62, pages 293–319, 1999. 44

[24] R. Botchen, D. Weiskopf, and T. Ertl. Texture-Based Visualization of Uncertainty in Flow Fields. In *VIS '05: Proceedings of the 16th IEEE Visualization*, pages 647–654. IEEE, 2005. 140

[25] S. Breslav, K. Szerszen, L. Markosian, P. Barla, and J. Thollot. Dynamic 2D Patterns for Shading 3D Scenes. *SIGGRAPH '07: ACM Transaction on Graphics*, 26(3):20, 2007. 122

[26] T. M. Breuel et al. OCRopus (tm): Open Source Document Analysis and OCR System. http://sites.google.com/site/ocropus, last access: 13-Sep-2010. 42

[27] P. Brown. The stick-e document: a framework for creating context-aware applications. In *IFIP Proceedings of Electronic Publishing*, pages 259–272, 1996. 24

[28] M. Broxvall, D. Radel, Y. Perret, P. Krueckel, S. Listopad, and A. Pollak. Trackballs. http://trackballs.sourceforge.net/, last access: 13-Sep-2010. 92

[29] S. Burigat and L. Chittaro. Location-aware visualization of VRML models in GPS-based mobile guides. In *Web3D '05: Proceedings of the tenth International Conference on 3D Web Technology*, pages 57–64. ACM Press, 2005. 68

[30] A. Butz, J. Baus, and A. Kruger. Augmenting Buildings with Infrared Information. In *ISAR '00: Proceedings of the IEEE and ACM International Symposium on Augmented Reality*, pages 93–96, 2000. 153

[31] A. Butz, J. Baus, A. Krüger, and M. Lohse. A Hybrid Indoor Navigation System. In *IUI '01: Proceedings of the 6th international conference on Intelligent user interfaces*, pages 25–32. ACM, 2001. 155

[32] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94: Proceedings of the 1994 Symposium on Volume Visualization*, pages 91–98. ACM, 1994. 82

[33] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986. 117

[34] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350 – 355, 1978. 108

[35] G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Department of Computer Science, Dartmouth College, 2000. 19, 20, 37

[36] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In *CHI '00: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 17–24. ACM, 2000. 68

[37] E. M. Coelho, B. MacIntyre, and S. J. Julier. OSGAR: A Scene Graph with Uncertain Transformations. In *ISMAR '04: Proceedings of the 3rd IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 6–15, 2004. 159

[38] Collaborative Research Centre (SFB627). Nexus: Spatial world models for mobile context-aware applications. http://www.nexus.uni-stuttgart.de, last access: 13-Sep-2010. 19, 20, 23, 151, 159

[39] P. Coschurba, U. Kubach, and A. Leonhardi. Research issues in developing a platform for spatial-aware applications. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 153–158. ACM Press, 2000. 23

[40] T. J. Cullip and U. Neumann. Accelerating Volume Reconstruction with 3D Texture Hardware. Technical Report TR93-027, University of North Carolina at Chapel Hill, 1993. 133

[41] N. Davies, K. Cheverst, and K. Mitchell. The GUIDE Project: Context-Sensitive Mobile Multimedia Support for City Visitors. http://www.guide.lancs.ac.uk, last access: 13-Sep-2010. 20

[42] M. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt. The triangle processor and normal vector shader: a VLSI system for high performance graphics. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 21–30. ACM, 1988. 59

[43] T. Delmarcelle and L. Hesselink. Visualizing second-order tensor fields with hyperstreamlines. *IEEE Computer Graphics and Application*, 13(4):25–33, 1993. 140

[44] P. Deutsch. DEFLATE compressed data format specification version 1.3. RFC1951: http://www.ietf.org/rfc/rfc1951.txt, 1996. 86

[45] R. W. DeVaul and S. Dunn. MIThril – The Context Aware Cell Phone Project. http://www.media.mit.edu/wearables/mithril, last access: 13-Sep-2010. 20

[46] A. K. Dey. Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7, 2001. 19

[47] A. K. Dey. The Context Toolkit. http://www.cs.cmu.edu/∼anind/context.html, last access: 13-Sep-2010. 24, 70, 151

[48] A. K. Dey and G. D. Abowd. Towards a better Understanding of Context and Context-Awareness. Technical Report GIT-GVU-99-22, Georgia Institute of Technology, 1999. 19

[49] M. Dhome, M. Richetin, J.-T. Lapresté, and G. Rives. Determination of the Attitude of 3D Objects from a Single Perspective View. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(12):1265–1278, 1989. 50, 51

[50] J. Diepstraten. *Interactive visualization methods for mobile device applications*. PhD thesis, University of Stuttgart, 2006. 84, 86

[51] J. Diepstraten and M. Eissele. In-Depth Performance Analyses of DirectX9 Shading Hardware concerning Pixel Shader and Texture Performance. In *Shader X3*, pages 523–544. Charles River Media, 2004. 57

[52] A. Doi and A. Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Institute of Electronics, Information and Communications Engineers Transactions*, E-74(1):214–224, 1991. 133

[53] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994. 135

[54] M. Eissele and J. Diepstraten. *GPU Performance of DirectX9 Per-Fragment Operations Revisited*, pages 541–560. Shader X4: Advanced Rendering with DirectX and OpenGL. Charles River Media, 2006. 57, 108

[55] M. Eissele and T. Ertl. Mobile Navigation and Augmentation utilizing Real-World Text. In *Mensch und Computer 2007, Workshop on Nomadic and Wearable Computing 2007*, pages 121–124, 2007. 38, 154

[56] M. Eissele, N. Hönle, T. Schwarz, S. Volz, M. Kada, L. Jendoubi, and S. Bürklen. 3D-Daten in GML 3.0. Technical Report 2006/03, SFB 627 Bericht, 2006. 28, 30

[57] M. Eissele, M. Kreiser, and T. Ertl. Context-Controlled Flow Visualization in Augmented Reality. In *GI '08: Proceedings of Graphics Interface*, pages 89–96. ACM Press, 2008. 83, 103, 140, 145, 159, 160

[58] M. Eissele, H. Sanftmann, and D. Weiskopf. Interactively Refining Object-Recognition System. *Journal of WSCG*, 17(1-3):1–8, 2009. 42, 43

[59] M. Eissele, O. Siemoneit, and T. Ertl. Transition of Mixed, Virtual, and Augmented Reality in Smart Production Environments - An Interdisciplinary View. In *RAM '06: Proceedings of the IEEE Conference on Robotics, Automation, and Mechatronics*, pages 1–6, 2006. 64, 99

[60] M. Eissele, S. Stegmaier, D. Weiskopf, and T. Ertl. Orientation as an additional User Interface in Mixed-Reality Environments. In *1. Workshop Erweiterte und Virtuelle Realität*, pages 79–90. GI-Fachgruppe AR/VR, 2004. 34, 35, 88, 99

[61] M. Eissele, D. Weiskopf, and T. Ertl. Frame-to-Frame Coherent Halftoning in Image Space. In *TPCG '04: Proceedings of Theory and Practice of Computer Graphics*, pages 188–195, 2004. 122

[62] M. Eissele, D. Weiskopf, and T. Ertl. The G^2-Buffer Framework. In *SimVis '04: Proceedings of the 15th Conference on Simulation and Visualisation*, pages 287–298, 2004. 59, 60

[63] M. Eissele, D. Weiskopf, and T. Ertl. Interactive Context-Aware Visualization for Mobile Devices. In *SG '09: Proceedings of Smart Graphics*, pages 167–178, 2009. 68, 166

[64] Ekahau. Real Time Location System (RTLS). http://www.ekahau.com/, last access: 13-Sep-2010. 32

[65] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 9–16. ACM, 2001. 135

[66] F. Hohl and U. Kubach and A. Leonhardi and K. Rothermel and M. Schwehm. Next Century Challenges: Nexus - An Open Global Infrastructure for Spatial-Aware Applications. In *MobiCom '99: Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 249–255. ACM Press, 1999. 37

[67] M. Falk and D. Weiskopf. Output-Sensitive 3D Line Integral Convolution. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):820–834, 2008. 140

[68] C. Faloutsos and K.-I. Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 163–174. ACM, 1995. 130

[69] K. Fast, T. Gifford, and R. Yancey. Virtual training for welding. In *ISMAR '04: Proceedings of the 3rd IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 298–299, 2004. 99

[70] J. Ferraiolo et al. Scalable Vector Graphics (SVG) 1.0 Specification. http://www.w3.org/TR/SVG10, last access: 13-Sep-2010. 60

[71] S. Ferreira, V. Garin, and B. Gosselin. A Text Detection Technique Applied in the Framework of a Mobile Camera-Based Application. In *CBDAR '05: Proceedings of Camera-based Document Analysis and Recognition (poster), Workshop of ICDAR*, pages 133–135, 2005. 41

[72] Firemint. Real Racing. http://www.firemint.com/realracing, last access: 13-Sep-2010. 37

[73] S. Floyd, M. Allman, A. Jain, and P. Sarolahti. Quick-Start for TCP and IP, IETF RFC 4782 (experimental). http://www.icir.org/floyd/quickstart.html, last access: 13-Sep-2010. 87

[74] B. Freudenberg, M. Masuch, and T. Strothotte. Real-time halftoning: A primitive for non-photorealistic shading. In *Proceedings of the 13th Eurographics Workshop on Rendering Techniques*, pages 227–231, 2002. 125

[75] D. Fritsch. Virtual cities and landscape models - what has photogrammetry to offer? In *Proceedings of the 47th Photogrammetric Week '99*, pages 3–14, 1999. 132

[76] G. W. Furnas. Generalized fisheye views. In *CHI '86: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 16–23. ACM Press, 1986. 98

[77] D. Garlan, D. P. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31, 2002. 20

[78] M. D. Garris, S. A. Janet, and W. W. Klein. Federal register document image database. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 3651 of *Document Recognition and Retrieval*, pages 97–108, 1999. 41, 42

[79] O. Gilson, N. Silva, P. W. Grant, and M. Chen. From web data to visualization via ontology mapping. *Computer Graphics Forum*, 27(3):959–966, 2008. 67

[80] Google Inc. Google Earth. http://earth.google.com/, last access: 13-Sep-2010. 28, 86, 169

[81] Google Inc. Google Maps. http://maps.google.com/, last access: 13-Sep-2010. 68, 154

[82] M. Grabner, H. Grabner, and H. Bischof. Fast visual object identification and categorization. In *NIPS '05: Proceedings of the Neural Information Processing Systems Workshop on Interclass Transfer*, pages 1–8, 2005. 43

[83] S. Green, D. Salesin, S. Schofield, A. Hertzmann, P. Litwinowicz, A. Gooch, C. Curtis, and B. Gooch. *Non-Photorealistic Rendering*. SIGGRAPH '99 Course Notes, 1999. 123

[84] T. Gu, H. K. Pung, and D. Q. Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1–18, 2005. 24

[85] C. Gutwin and C. Fedak. Interacting with big interfaces on small screens: a comparison of fisheye, zoom, and panning techniques. In *GI '04: Proceedings of Graphics Interface 2004*, pages 145–152. Canadian Human-Computer Communications Society, 2004. 104

[86] R. B. Haber and D. A. McNabb. Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. In *Visualization in Scientific Computing*, pages 74–93. IEEE Computer Scociety Press, 1990. 62, 177

[87] C. D. Hansen and C. R. Johnson. *The Visualization Handbook*. Academic Press, Inc., 2004. 128, 135, 140

[88] R. L. Harris. *Information Graphics: A Comprehensive Illustrated Reference*. Oxford University Press, Inc., 1999. 130

[89] B. L. Harrison, K. P. Fishkin, A. Gujar, C. Mochon, and R. Want. Squeeze Me, Hold Me, Tilt Me! An Exploration of Manipulative User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 17–24. ACM Press, 1998. 34, 89, 93

[90] H. Hauser. Generalizing focus+context visualization. In *Proceedings of the Dagstuhl 2003 Seminar on Scientific Visualization: The Visual Extraction of Knowledge from Data*, pages 305–327. Springer, 2005. 98

[91] S. Hemminger. Network emulation with netem. In *LCA '05: Proceedings of the 6th Australian National Linux Conference*. (electronic proceedings), 2005. 87

[92] I. Herman, I. C. Society, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6:24–43, 2000. 128

[93] A. Hertzmann and K. Perlin. Painterly rendering for video and interaction. In *NPAR 2000: First International Symposium on Non-Photorealistic Animation and Rendering*, pages 7–12, 2000. 123

[94] K. Hinckley, J. Pierce, M. Sinclair, and E. Horvitz. Sensing Techniques for Mobile Interaction. In *UIST '00: Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, pages 91–100. ACM Press, 2000. 34

[95] M. Hopf and T. Ertl. Hierarchical Splatting of Scattered Data. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003*, pages 443–440. IEEE Computer Society, 2003. 131

[96] K. Hornbaek, B. B. Bederson, and C. Plaisant. Navigation patterns and usability of zoomable user interfaces with and without an overview. *ACM Transactions on Computer-Human Interaction*, 9(4):362–389, 2002. 104

[97] W.-H. Hsu, J. Mei, C. D. Correa, and K.-L. Ma. Depicting time evolving flow with illustrative visualization techniques. In *ArtsIT '09: Proceedings of the International Conference on Arts and Technology*, pages 136–147, 2009. 140

[98] H. Hu and G. de Haan. Simultaneous Coding Artifact Reduction and Sharpness Enhancement. In *ICCE '07: International Conference on Consumer Electronics, Digest of Technical Papers*, pages 213–214. IEEE Computer Society Press, 2007. 115

[99] A. Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(4):69–91, 1985. 130

[100] InterSense Inc. InterSense InertiaCube^2 Specifications. http://www.intersense.com/uploadedFiles/Products/IC2+_datasheet_0908.pdf, last access: 13-Sep-2010. 33

[101] InterSense Inc. InterSense IS900. http://www.intersense.com/uploadedFiles/Products/IS900.pdf, last access: 13-Sep-2010. 32, 152

[102] L. Jagannathan and C. V. Jawahar. Crosslingual access of textual information using camera phones. In *Proceedings of the International Conference on Cognition and Recognition*, pages 655–660. Allied Publishers, 2005. 38, 156

[103] L. Jagannathan and C. V. Jawahar. Perspective correction methods for camera based document analysis. In *CBDAR'05: Proceedings of the 1st International Workshop on Camera-based Document Analysis and Recognition*, pages 148–154, 2005. 41

[104] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Lagrangian-Eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):211–222, 2002. 124

[105] B. Jobard and W. Lefer. Multiresolution flow visualization. In *WSCG (Posters)*, pages 34–35, 2001. 140

[106] E. Jung and K. Sato. A framework of context-sensitive visualization for user-centered interactive systems. In *10th International Conference on User Modeling*, pages 423–427. Springer, 2005. 68

[107] M. Kada, S. Roettger, K. Weiss, T. Ertl, and D. Fritsch. Real-time visualisation of urban landscapes using open-source software. In *ACRS '03: Proceedings of the 24th Asian Conference on Remote Sensing*. (electronic proceedings), 2003. 71

[108] J. T. Kajiya and B. P. Von Herzen. Ray tracing volume densities. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 165–174. ACM, 1984. 133

[109] H. Kato and M. Billinghurst. Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System. In *IWAR '99: Proceedings of the 2nd International Workshop on Augmented Reality*, pages 85–94, 1999. 38

[110] H. Kato and M. Billinghurst. ARToolKit. http://artoolkit.sourceforge.net/, last access: 13-Sep-2010. 34, 38, 40, 159

[111] Khronos Group. Collada specification. http://www.khronos.org/collada/, last access: 13-Sep-2010. 28

[112] Khronos Group. OpenCL - The open standard for parallel programming of heterogeneous systems. http://www.khronos.org/opencl, last access: 13-Sep-2010. 59, 178

[113] Khronos Group. OpenVG - The Standard for Vector Graphics Acceleration. http://www.khronos.org/openvg, last access: 13-Sep-2010. 61

[114] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 79–86. ACM, 1998. 116, 118

[115] Kitware Inc. The Visualization Toolkit (VTK). http://www.vtk.org/, last access: 13-Sep-2010. 71

[116] T. Klein, M. Eissele, D. Weiskopf, and T. Ertl. Simulation, Modelling and Rendering of Incompressible Fluids in Real Time. In *VMV '03: Workshop on Vision, Modelling, and Visualization*, pages 365–373. infix, 2003. 58, 137

[117] T. Klein, S. Stegmaier, and T. Ertl. Hardware-accelerated Reconstruction of Polygonal Isosurface Representations on Unstructured Grids. In *Proceedings of Pacific Graphics '04*, pages 186–195, 2004. 133

[118] M. Koga, R. Mine, T. Kameyama, T. Takahashi, M. Yamazaki, and T. Yamaguchi. Camera-based Kanji OCR for mobile-phones: practical issues. In *ICDAR'05: Proceedings of the 8th International Conference on Document Analysis and Recognition*, volume 2, pages 635–639, 2005. 38, 156

[119] T. Kondo and K. Kawaguchi. Adaptive dynamic range encoding method and apparatus. US-patent: no. 5,444,487, 1995. 112

[120] T. Kondo, Y. Node, T. Fujiwara, and Y. Okumura. Picture conversion apparatus, picture conversion method, learning apparatus and learning method. US-patent: no. 6,323,905, 2001. 111, 112

[121] R. Kosara, S. Miksch, and H. Hauser. Semantic depth of field. In *INFOVIS '01: Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, pages 97–104. IEEE Computer Society, 2001. 135

[122] M. Kraus, M. Eissele, and M. Strengert. GPU-Based Edge Directed Image Interpolation. In *Image Analysis (Proceedings of SCIA 2007)*, volume 4522 of *Lecture Notes in Computer Science*, pages 532–541. Springer, 2007. 116, 118

[123] M. Krause, C. Linnhoff-Popien, and M. Strassberger. Concurrent inference on high level context using alternative context construction trees. In *ICAS '07: Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, pages 7–12. IEEE Computer Society, 2007. 19

[124] J. Krüger, J. Schneider, and R. Westermann. ClearView: An interactive context preserving hotspot visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):941–948, 2006. 135

[125] A. Leonhardi and K. Rothermel. Architecture of a Large-scale Location Service. Technical Report 2001/01, Universität Stuttgart, 2001. 26

[126] X. Li and M. Orchard. New edge-directed interpolation. *IEEE Transactions on Image Processing*, 10(10):1521–1527, 2001. 115

[127] D. Liu, M. Allman, S. Jin, and L. Wang. Congestion control without a startup phase. In *PFLDnet '07: Proceedings of the International Workshop on Protocols for Fast Long-Distance Networks*. (electronic proceedings), 2007. 87

[128] Z. Liu, R. Moorhead, and J. Groner. An advanced evenly-spaced streamline placement algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):965–972, 2006. 140

[129] W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21(4):163–169, 1987. 133

[130] D. G. Lowe. Three-dimensional object recognition from single two-dimensional images. In *Artificial Intelligence*, volume 31, pages 355–395. Elsevier, 1987. 50, 51

[131] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. 38

[132] D. Lucke, E. Westkämper, M. Eissele, T. Ertl, and O. Siemoneit. Privacy-Preserving Self-Localization Techniques in Next Generation Manufacturing - An Interdisciplinary View on the Vision and Implementation of Smart Factories. In *ICARCV '08:Proceedings of the 10th International Conference on Control, Automation, Robotics and Vision*, pages 1183–1188, 2008. 38, 39, 153, 154

[133] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, 1986. 95

[134] D. Marr and E. C. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B*, 207:187–217, 1980. 117

[135] P. Martz. *OpenSceneGraph Quick Start Guide*. Skew Matrix Software LLC, 2007. http://www.osgbooks.com/books, last access: 13-Sep-2010. 62

[136] R. Mehrotra, W. Grosky, and F. Kung. Decision-tree based two-dimensional object recognition. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 1380–1383, 1988. 43

[137] B. J. Meier. Painterly rendering for animation. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 477–484. ACM, 1996. 121

[138] E. Méndez, D. Kalkofen, and D. Schmalstieg. Interactive context-driven visualization tools for augmented reality. In *ISMAR '06: Proceedings of IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 209–218, 2006. 68

[139] Microsoft Corporation. Direct3D 11 Technical Preview. http://www.microsoft.com/directx, last access: 13-Sep-2010. 58, 59, 178

[140] Microsoft Corporation. Microsoft Virtual Earth. http://earth.live.com/, last access: 13-Sep-2010. 86, 169, 171

[141] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino. Augmented reality: A class of displays on the reality-virtuality continuum. In *SPIE Vol. 2351, Telemanipulator and Telepresence Technologies*, pages 282–292, 1994. 64

[142] S. Möser, P. Degener, R. Wahl, and R. Klein. Context aware terrain visualization for wayfinding and navigation. *Computer Graphics Forum*, 27(7):1853–1860, 2008. 21

[143] J. Nausedat. Hardware-Based Techniques for 2D Video Processing. University of Stuttgart, Department of Computer Science, diploma thesis Nr. 2534, 2007. 111, 112, 114

[144] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965. 114

[145] D. Nicklas and B. Mitschang. The nexus augmented world model: An extensible approach for mobile, spatially-aware applications. In *OOIS '01: Proceedings of the 7th International Conference on Object-Oriented Information Systems*, pages 392–401, 2001. 74, 75, 171

[146] Nintendo Inc. Nintendo Wii Gaming Console. http://www.nintendo.com/wii, last access: 13-Sep-2010. 92

[147] Y. Noimark and D. Cohen-Or. Streaming Scenes to MPEG-4 Video-Enabled Devices. *Computer Graphics and Applications, IEEE*, 23(1):58–64, 2003. 86

[148] NVIDIA Corporation. CUDA: Compute Unified Device Architecture. http://www.nvidia.com/cuda, last access: 13-Sep-2010. 59, 178

[149] NVIDIA Corporation. Next Generation CUDA Compute Architecture: Fermi. http://www.nvidia.com/object/fermi_architecture.html, last access: 13-Sep-2010. 178

[150] NVIDIA Corporation. NVSG - The Nvidia Scene Graph API. http://developer.nvidia.com/object/nvsg_home.html, last access: 13-Sep-2010. 61

[151] A. Oliva and A. Torralba. The role of context in object recognition. *Trends in Cognitive Sciences*, 11(12):520–527, 2007. 20

[152] Open GIS Consortium Inc. OpenGIS City Geography Markup Language (CityGML) Encoding Standard. http://www.opengeospatial.org/standards/citygml, 2008. 28

[153] Open GIS Consortium Inc. OpenGIS Geography Markup Language (GML) Implementation Specification. http://www.opengis.org/docs/02023r4.pdf, 2008. 28

[154] R. Oppermann and M. Specht. A context-sensitive nomadic exhibition guide. In *HUC '00: Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 127–142. Springer, 2000. 154

[155] R. Osfield, D. Burns, et al. OpenSceneGraph. http://www.openscenegraph.org, 2007. 61, 62, 94, 100

[156] K. Partridge, S. Chatterjee, V. Sazawal, G. Borriello, and R. Want. TiltType: Accelerometer-Supported Text Entry for Very Small Devices. In *UIST '02: Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology*, pages 201–204. ACM Press, 2002. 34

[157] J. Pascoe. The stick-e note architecture: extending the interface beyond the user. In *IUI '97: Proceedings of the 2nd international conference on Intelligent user interfaces*, pages 261–264. ACM, 1997. 24

[158] K. Perlin and D. Fox. Pad: an alternative approach to the computer interface. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 57–64. ACM, 1993. 104

[159] M. Peter. Presentation and evaluation of inconsistencies in multiply represented 3d building models. In *QuaCon'09: Proceedings of the 1st international conference on Quality of context*, pages 156–163. Springer-Verlag, 2009. 142

[160] M. Peter, N. Haala, and D. Fritsch. Preserving ground plan and facade lines for 3d building generalization. In *ISPRS '08: 21st Congress of the International Society for Photogrammetry and Remote Sensing*, volume XXXVII of *Commission II*, pages 481–485, 2008. 142, 173

[161] H. Piringer, R. Kosara, and H. Hauser. Interactive focus+context visualization with linked 2d/3d scatterplots. In *CMV '04: Proceedings of the Second International Conference on Coordinated & Multiple Views in Exploratory Visualization*, pages 49–60. IEEE Computer Society, 2004. 135

[162] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-time hatching. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 581–586. ACM, 2001. 122

[163] H. K. Pung and T. Gu. Context-aware middleware services and programming support for sentient computing. http://telehealth.i2r.a-star.edu.sg/projects/projects.html, last access: 13-Sep-2010. 24, 151

[164] J. Raskin. The Humane Environment (THE). http://sourceforge.net/projects/humane/, last access: 13-Sep-2010. 104

[165] M. Reinhold, F. Deinzer, J. Denzler, D. Paulus, and J. Pösl. Active Appearance-Based Object Recognition Using Viewpoint Selection. In *VMV '00: Workshop on Vision, Modeling, and Visualization*, pages 105–112, 2000. 44

[166] G. Reitmayr and D. Schmalstieg. Flexible parameterization of scene graphs. In *VR '05: IEEE Virtual Reality Conference*, pages 51–58, 2005. 68

[167] J. Rekimoto. Tilting Operations for Small Screen Interfaces. In *UIST '96: Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*, pages 167–168, 1996. 34, 35, 88

[168] J. Rekimoto and K. Nagao. The world through the computer: computer augmented interaction with real world environments. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 29–36. ACM, 1995. 68

[169] P. J. Rhodes, R. S. Laramee, R. D. Bergeron, and T. M. Sparr. Uncertainty Visualization Methods in Isosurface Rendering. In *Proceedings of Eurographics 2003 Short Papers*, pages 83–88, 2003. 135

[170] F. Rößler, R. P. Botchen, and T. Ertl. Dynamic Shader Generation for Flexible Multi-Volume Visualization. In *PacificVis '08: Proceedings of IEEE Pacific Visualization Symposium*, pages 17–24, 2008. 82

[171] A. R. Rondeau et al. Semapedia.org Project. http://www.semapedia.org/, last access: 13-Sep-2010. 38

[172] M. Rotard, M. Eissele, R. Van Putten, and T. Ertl. Zoomable User Interfaces in SVG. In *SVG Open 2007*. http://www.svgopen.org/ (electronic proceedings), 2007. 104

[173] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003. 43

[174] T. Saito and T. Takahashi. Comprehensible rendering of 3-D shapes. In *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, volume 24, pages 197–206, 1990. 59, 148

[175] D. Salber, A. K. Dey, and G. D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *CHI '99: Proceedings of the 1999 Conference on Human Factors in Computing Systems*, pages 434–441, 1999. 24, 70

[176] J. Savolainen, H. Hirvola, and S. Iraji. EPC UHF RFID Reader: Mobile Phone Integration and Services. In *CCNC '09: 6th IEEE Consumer Communications and Networking Conference*, pages 1–5, 2009. 33

[177] M. Scharf, M. Eissele, C. Mueller, and T. Ertl. Speeding up the 3D Web: A Case for Fast Startup Congestion Control. In *PFLDnet '09: Proceedings of the 7th International Workshop on Protocols for Fast Long-Distance Networks*. (electronic proceedings), 2009. 86

[178] G. Scheuermann, T. Bobach, H. Hagen, K. Mahrous, B. Hamann, K. I. Joy, and W. Kollmann. A tetrahedra-based stream surface algorithm. In *VIS '01: Proceedings of the IEEE Visualization Conference*, pages 151–158. IEEE Computer Society, 2001. 140

[179] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE Computer Society, 1994. 18, 19

[180] A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to Context than Location. *Computers and Graphics*, 23(6):893–901, 1999. 34, 37, 88, 91

[181] M. Schnaider, B. Schwald, H. Seibert, and T. Weller. Medarpa - A Medical Augmented Reality System for Minimal-Invasive Interventions. In *Proceedings of Medicine Meets Virtual Reality 2003*, pages 312–314. IOS Press, 2003. 92

[182] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit, first edition*. Kitware Inc., 1993. 71

[183] M. Segal and K. Akeley. The OpenGL Graphics System: A Specification (Ver: 2.0). http://www.opengl.org/, last access: 13-Sep-2010. 58, 71

[184] H. Senay and E. Ignatius. A knowledge-based system for visualization design. *IEEE Compututer Graphics and Application*, 14(6):36–47, 1994. 67

[185] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524. ACM, 1968. 131

[186] F. Shibata, T. Hashimoto, K. Furuno, A. Kimura, and H. Tamura. Scalable architecture and content description language for mobile mixed reality systems. In *ICAT '06: 16th International Conference on Artificial Reality and Telexistence*, volume 4282 of *Lecture Notes in Computer Science*, pages 122–131. Springer, 2006. 68

[187] J. Small, A. Smailagic, and D. Siewiorek. Determining User Location for Context-Aware Computing through the Use of a Wireless LAN Infrastructure. Technical report, ICES CMU, 2000. 154

[188] S. Stegmaier, J. Diepstraten, M. Weiler, and T. Ertl. Widening the Remote Visualization Bottleneck. In *ISPA '03: Procceedings of the 3rd International Syposium on Images and Signal Processing and Analysis*, pages 174–179. IEEE Press, 2003. 86

[189] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl. A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In *Volume Graphics '05*, pages 187–195, 2005. 69, 133

[190] P. S. Strauss. IRIS Inventor, a 3D graphics toolkit. *SIGPLAN Notices*, 28(10):192–200, 1993. 61

[191] M. Strengert, M. Kraus, and T. Ertl. Pyramid Methods in GPU-Based Image Processing. In *VMV '06: Workshop on Vision, Modelling, and Visualization*, pages 169–176, 2006. 108, 110, 118

[192] D. Tan, I. Poupyrev, M. Billinghurst, H. Kato, H. Regenbrecht, and N. Tetsutani. On-demand, in-place help for augmented reality environments. In *Ubicomp '01: Proceedings of the International Conference on Ubiquitous Computing (short paper)*, 2001. 34

[193] Technalysis Inc. Passage/Flow: Cleanroom Flow Modeling. http://www.technalysis.com/clean_room_design.aspx, last access: 13-Sep-2010. 165

[194] X. Tricoche, G. Scheuermann, and H. Hagen. Vector and tensor field topology simplification on irregular grids. In *VISSYM '01: Proceedings of Data Visualization 2001*, pages 107–116, 2001. 140

[195] P. Tschirner, B. Hillers, and A. Graeser. A Concept for the Application of Augmented Reality in Manual Gas Metal Arc Welding. In *ISMAR '02: Proceedings the International Symposium on Mixed and Augmented Reality*, pages 257–258, 2002. 99

[196] Ubisense Ltd. Series 7000 system. http://www.ubisense.de/, last access: 13-Sep-2010. 32, 152

[197] C. Verplaetse. Inertial proprioceptive devices: Self-motion-sensing toys and tools. *IBM Systems Journal*, 35(3-4):639–650, 1996. 34

[198] P. Viola and M. J. Jones. Robust Real-Time Face Detection. *International Journal of Computer Vision*, 57(2):137–154, 2004. 43

[199] W3C. X3D Specification Schema. http://www.web3d.org/x3d/specifications/schema/, last access: 13-Sep-2010. 28

[200] Q. Wang and R. Ward. A new edge-directed image expansion scheme. *Proceedings of the International Conference on Image Processing*, 3:899–902, 2001. 115

[201] R. Want, A. Hopper, V. Falc ao, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, 1992. 20, 154

[202] T. Weinkauf, H. Theisel, K. Shi, H.-C. Hege, and H.-P. Seidel. Extracting higher order critical points and topological simplification of 3D vector fields. In *VIS '05: Proceedings of IEEE Visualization*, pages 559–566, 2005. 140

[203] D. Weiskopf. *GPU-Based Interactive Visualization Techniques (Mathematics and Visualization)*. Springer, 2006. 69

[204] D. Weiskopf, G. Erlebacher, M. Hopf, and T. Ertl. Hardware-accelerated Lagrangian-Eulerian texture advection for 2D flow visualization. In *VMV '02: Workshop on Vision, Modeling, and Visualization*, pages 77–84, 2002. 125

[205] D. Weiskopf and T. Ertl. A hybrid physical/device-space approach for spatio-temporally coherent interactive texture advection on curved surfaces. In *Proceedings of Graphics Interface*, pages 263–270. IEEE Computer Society, 2004. 139

[206] E. Westkämper and L. Jendoubi. Smart Factories - Manufacturing Environments and Systems of the Future. In *CIRP '03: The 36th International CIRP Seminar on Manufacturing Systems*, pages 13–16, 2003. 20, 164, 165

[207] E. Westkämper, L. Jendoubi, M. Eissele, and T. Ertl. Smart Factory - Bridging the gap between digital planning and reality. In *CIRP '05: The 38th International CIRP Seminar on Manufacturing Systems*, pages 1–6, 2005. 20, 99, 152, 163

[208] E. Westkämper, L. Jendoubi, M. Eissele, T. Ertl, and J. Niemann. Smart Factories - Intelligent Manufacturing Environments. *Journal of Machine Engineering*, 5(114-122):114–122, 2005. 99, 164

[209] S. White, S. Feiner, and J. Kopylec. Virtual vouchers: Prototyping a mobile augmented reality user interface for botanical species identification. In *3DUI '06: Proceedings of the 3D User Interfaces*, pages 119–126. IEEE Computer Society, 2006. 69

[210] O. Wilson, A. VanGelder, and J. Wilhelms. Direct Volume Rendering via 3D Textures. Technical Report UCSC-CRL-94-19, University of California, 1994. 133

[211] H. Wright. *Introduction to Scientific Visualization*. Springer, 2007. 128, 135, 140

[212] W. Yeung and J. Ng. Wireless LAN Positioning based on Received Signal Strength from Mobile device and Access Points. In *RTCSA '07: 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 131–137, 2007. 32

[213] M. Zöckler, D. Stalling, and H.-C. Hege. Interactive visualization of 3D-vector fields using illuminated stream lines. In *VIS '96: Proceedings of the 7th conference on Visualization 1996*, pages 107–113. IEEE Computer Society Press, 1996. 83, 138, 145