# Rendering Methods for Augmented Reality

**Dissertation**
der Fakultät für Informations- und Kognitionswissenschaften
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
**Dipl.-Inform. Jan T. Fischer**
aus Mainz

**Tübingen**
**2006**

| | |
|---|---|
| Tag der mündlichen Qualifikation: | 14.06.2006 |
| Dekan: | Prof. Dr. rer. soc. Michael Diehl |
| 1. Berichterstatter: | Prof. Dr.-Ing. Dr.-Ing. E.h. Wolfgang Straßer |
| 2. Berichterstatter: | Prof. Dr. rer. nat. Bernd Fröhlich<br>(Bauhaus-Universität Weimar) |
| 3. Berichterstatter: | PD Dr. rer. nat. Dirk Bartz |

## Erklärung

Hiermit erkläre ich, dass ich die Arbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die im Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben der Quellen als Entlehnung kenntlich gemacht worden sind.

Tübingen, Februar 2006 *Jan Fischer*

## Zusammenfassung

In den letzten Jahren hat sich die *Erweiterte Realität* (englisch: *Augmented Reality*) zu einer vielversprechenden und schnell an Bedeutung gewinnenden Anwendung der Computergraphik entwickelt. Augmented-Reality-Systeme kombinieren vom Computer erzeugte graphische Darstellungen mit der Ansicht der realen Welt. Mehrere zentrale Problemstellungen lassen sich im Bereich der Augmented Reality identifizieren. Dabei handelt es sich um die Entwicklung spezieller Anzeigegeräte, das Kamera-Tracking, den Systementwurf, die Benutzerinteraktion und Darstellungsverfahren. Während sich der Großteil vorhergehender Arbeiten mit den Problemen des Systementwurfs, des Kamera-Trackings und mit Anwendungen der Augmented Reality befasste, liegt der Schwerpunkt dieser Doktorarbeit auf dem bislang relativ wenig beachteten Thema der Darstellungstechniken.

Im ersten Teil dieser Doktorarbeit (Kapitel 2) wird ein neu entwickeltes System für die medizinische Augmented Reality vorgestellt [22]. Das *ARGUS*-Projekt ist ein neues Augmented-Reality-System, das auf einem kommerziellen intraoperativen Navigationsgerät basiert, wodurch ein Einsatz in der klinischen Praxis erleichtert werden könnte. Mehrere Erweiterungen des grundlegenden Projekts werden beschrieben, darunter ein hybrides Tracking-Verfahren, eine Bibliothek für die Benutzerinteraktion und eine Methode für die Verdeckungsbehandlung. Letztere macht es möglich, die Verdeckung graphischer Objekte durch die Patientenanatomie korrekt darzustellen, was zu einer realistischeren und verständlicheren Ausgabe führt. Dieser Algorithmus ist eine der fortgeschrittenen Darstellungstechniken für Augmented Reality, die im Rahmen dieser Doktorarbeit entwickelt wurden.

Der zweite Teil dieser Doktorarbeit, Kapitel 3, stellt das neue Konzept der *Stylized Augmented Reality* vor. Dabei werden künstlerische oder illustrative Stilisierungsmethoden auf Augmented-Reality-Videoströme angewendet [15]. Da die gleiche Art der Stilisierung auf virtuelle und reale Bildelemente angewendet wird, nimmt ihre Unterscheidbarkeit stark ab. Auf diese Art und Weise wird ein neuartiges Augmented-Reality-Erlebnis vermittelt, und möglicherweise wird eine bessere Immersion erzeugt. Echtzeitverfahren für die Cartoon-artige und malerische Stilisierung von Augmented-Reality-Bildern werden beschrieben. Auch der Einsatz programmierbarer Graphikhardware zu diesem Zweck wurde untersucht. Zudem werden die Ergebnisse einer psychophysikalischen Studie über die Unterscheidbarkeit virtueller Objekte in der Stylized Augmented Reality präsentiert.

Zu Beginn von Kapitel 4, das den dritten Teil dieser Doktorarbeit darstellt, wird eine neue Methode der illustrativen Visualisierung beschrieben. Dieser neue Darstellungsalgorithmus für Isoflächen und polygonale Modelle erzeugt die illustrative Abbildung einer Oberfläche und dahinter versteckter Strukturen [12]. Die Methode wurde für die programmierbaren Architekturen moderner Graphikhardware entworfen und kann komplexe Modelle in Echtzeit darstellen. Eine Erweiterung dieses neu entwickelten Darstellungsstils wurde auch auf Augmented-Reality-Videoströme angewendet. Diese Entwicklung stellt eine weitere Realisierung des Konzepts der Stylized Augmented Reality dar.

## Abstract

Augmented reality (AR) has become a promising and fast-growing application of computer graphics over the course of the last years. Augmented reality systems overlay computer-generated graphical information over the view of the real world. Several main research challenges can be identified in the field of augmented reality. These are the design of advanced display devices (e.g., head-mounted displays), camera tracking, system design, user interaction, and rendering. While a major part of the previous work focused on the problems of system design, camera tracking, and applications of AR, this thesis puts a different emphasis on the relatively underrepresented aspect of rendering techniques. In this thesis, several novel methods for displaying augmented video streams are explored.

In the first part of this thesis in Chapter 2, the design and implementation of a novel system for medical augmented reality are discussed [22]. The *ARGUS* framework is a new augmented reality system based on a commercial surgical navigation device. Since it does not require any additional hardware components, a transition into the clinical practice can be facilitated. Several extensions of the basic framework are described, including a hybrid tracking scheme, a user interaction library, and a method for handling occlusion. The latter algorithm makes it possible to correctly render the occlusion of graphical objects by the anatomy of the patient, leading to a more realistic and easily comprehensible output. This approach is one of the advanced rendering methods for augmented reality investigated in the context of this thesis.

The second part of this thesis, Chapter 3, introduces the concept of *stylized augmented reality*, which applies artistic or illustrative stylization methods to augmented reality video streams [15]. Since the same type of stylization is applied to virtual and real scene elements, they become difficult to distinguish. This way, a novel augmented reality experience is created, and possibly even a better immersion. Real-time algorithms for cartoon-like and painterly brush stroke stylization of augmented video streams are described. The exploitation of programmable graphics hardware for this purpose is discussed. Moreover, the results of a psychophysical study on the discernability of virtual objects in stylized augmented reality are presented.

At the beginning of Chapter 4, which is the third part of this thesis, a novel illustrative visualization method is described. This new rendering algorithm for iso-surfaces and polygonal models generates an illustrative representation of a surface and structures hidden behind it [12]. The method is designed for the programmable rendering pipelines of modern graphics hardware and is capable of displaying complex models in real-time. An extension of this newly developed illustrative display style was also applied to augmented reality video streams. This system constitutes another realization of the concept of stylized augmented reality.

## Acknowledgements

This thesis was conducted during my occupation as research assistant in the group Visual Computing for Medicine (VCM) at the department Graphical-Interactive Systems (WSI/GRIS) of the University of Tübingen. The research described in this dissertation would not have been possible without the support and encouragement of many people. First and foremost, I would like to thank my advisors Prof. Dr.-Ing. Dr.-Ing. E.h. Wolfgang Straßer (head of the department WSI/GRIS) and PD Dr. Dirk Bartz (head of VCM). They provided me with a pleasant and productive work atmosphere and the opportunity to freely explore different research topics, even if these sometimes stretched the strict definition of my research grant a little. I would also like to thank Prof. Dr. Bernd Fröhlich, who agreed to be part of my graduation committee.

Another main reason for the good and fruitful environment were the colleagues with whom I worked together in the department. In particular, I would like to express my gratitude towards Ángel del Río for the inspiring scientific collaboration as well as countless pounds of great Spanish coffee. I would also like to thank Zein Salah, who is another member of the VCM group, for many interesting conversations. The colleagues who shared the office with me, Stefan Kimmerle and Bernhard Thomaszewski, deserve to be mentioned, if nothing else, for bearing with my curses and other types of monologues at the computer screen, their sense of humor, and having to cope with mine. Many other coworkers at the GRIS helped me to broaden my horizon with regard to science, the academic world, and through interesting and enjoyable common conference trips. To name only some of them, I would like to thank Andreas Schilling, Ralf Sondershaus, Peter Biber, Sven Fleck, Dirk Staneker, Michael Keckeisen, Markus Wacker, Michael Wand, Philipp Jenke, Urs Kanus, Gregor Wetekam, Johannes Mezger, Stefan Guthe, Michael Hauth, Johannes Görke, Stefan Gumhold, and Johannes Hirche. Of course, any scientific achievement would be impossible without the support and organization skills of the secretaries, who are often appropriately regarded as the persons who actually run the department. I would therefore like to acknowledge Constanze Christ, Adelheid Ebert, and Bernadette Graus for dealing with my constant requests for money for webcams and conference visits, and for making (almost) all of it possible.

Several students contributed to the implementation of some of the concepts presented in this thesis. Melissa Mekić, Christine Schaller, Martin Köbele, and Michael Eichler not only delivered very good student and Master theses, but advising them was also a gratifying and fun experience.

Finally, I would also like to thank the academic and industrial collaborators who helped to achieve the results presented in this thesis. As clinical partners in the VIRTUE research project, Jürgen Hoffmann, Carsten Westendorff, Frank Duffner, Dirk Freudenstein, and Marcos Tatagiba from the University Hospital Tübingen provided guidance with regard to potential medical applications of the newly developed techniques. The people at the BrainLAB company helped with the technical aspects of accessing the internal data of their image guided surgery system, and especially the very friendly support of Markus Neff, Ulrich Seifert, and Sven Flossmann deserves to be mentioned. A very interesting line of research was opened by the fruitful exchange of ideas with Douglas Cunningham and Christian Wallraven (Max-Planck-Institute of Biological Cybernetics), and their support during the psychopysical study described here was very much appreciated.

This thesis was funded by the research grant VIRTUE in the focus program on ”Medical Navigation and Robotics” (SPP 1124) of the German Research Foundation (DFG).

# CONTENTS

# LIST OF FIGURES

CHAPTER 1

# Introduction

## 1.1 Augmented Reality

In computer graphics, generated images and animations are usually presented to the user on dedicated display devices. These include conventional screens and video projectors, but also the conversion of digital images to classical media like movies and television. Moreover, a number of advanced display technologies for computer graphics has been developed in recent years. These sophisticated systems typically offer large perceived display areas, which sometimes surround the user, and stereoscopic rendering. Examples of advanced displays are head-mounted displays [89], rooms made up of back-projected screens (CAVEs) [73], table-like display setups [129], and domed projection systems [82]. Most of these more elaborate systems originate from virtual reality (VR) research [197]. They are normally used in high-end applications like scientific and medical visualization and industrial design, but also in entertainment settings.

When using such dedicated display devices, the attention of the observer is focused on the computer-generated images, limiting the perception of the real surroundings. For many scenarios, however, it is desirable or even necessary to provide the user with a view of the actual environment along with the artificial graphics. This is usually the case whenever an interactive display of virtual information is to be integrated into some real-life task. The necessity of such a combination of real and virtual image elements has lead to the development of *augmented reality* (AR). When one looks at the history of early computer graphics research, it becomes apparent that augmented reality is not an entirely novel concept. In 1968 one of the pioneers of computer graphics, Ivan E. Sutherland, described a setup which could well be regarded the first augmented reality system [191]:

As the observer moves his head, his point of view moves and rotates with respect to the room coordinate system. [...] Half-silvered mirrors in the prisms through which the user looks allow him to see both the images from the cathode ray tubes and objects in the room simultaneously. Thus displayed material can be made either to hang disembodied in space or coincide with maps, desk tops, walls, or the keys of a typewriter.

(Ivan E. Sutherland, *A Head-Mounted Three-Dimensional Display*)

In recent years, augmented reality has become an area of active research. These efforts have also produced a more detailed definition and classification of augmented reality. Azuma enumerates the following characteristics as essential for an AR system [43, 44]:

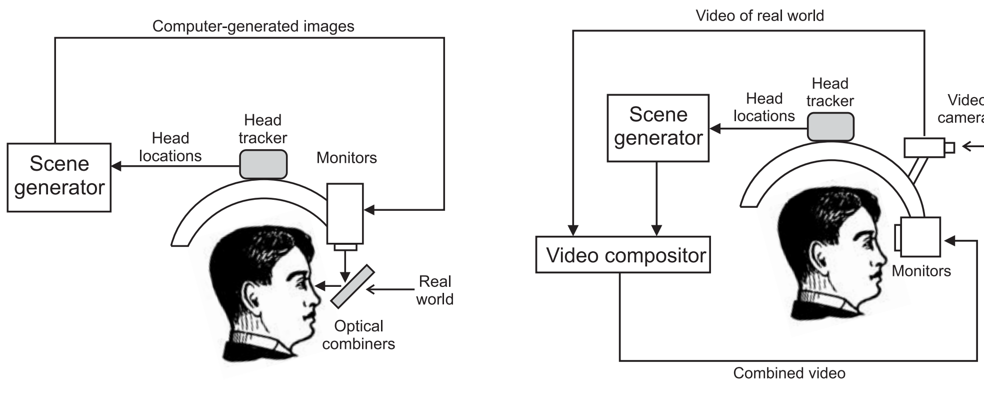
1. **Combination of real and virtual**. Artifically generated graphics and images of the real environment are combined. This is the main feature of augmented reality. It is not, however, a unique attribute of AR. Research trends in fields like ubiquitous computing [205] and wearable computing [143] also aim at providing the user with an overlay of information over the normal field of view (usually 2D graphics and text overlays). Therefore, the following two characteristics are necessary for a more precise definition of augmented reality.

2. **Real-time interactivity**. The virtual image elements presented to the user are generated in real-time, and their appearence is sensitive to user input and changes in the environment.

3. **Three-dimensional registration**. Graphical information in the AR environment has a correct spatial alignment relative to the actual surroundings of the user. In order to achieve such a useful registration, the head of the user or the camera used in the system has to be tracked. This consistent three-dimensional alignment is the distinguishing feature of augmented reality.

In his survey article, Azuma differentiates between two types of basic designs for an AR system [43]. *Optical see-through* approaches use optical combiners, which are placed in front of the user's eyes. These combiners are partially translucent, so that the real world remains directly visible. They are also partially reflective, so that the user can additionally see virtual images shown on a display device, e.g., a head-mounted monitor. In *video see-through* augmented reality, a video camera is used for recording images of the real environment (two cameras in the case of a stereo AR system). The acquired digital images then serve as background bitmaps, over which the virtual graphical objects are rendered. The resulting combined digital images are presented to the user on non-translucent display devices like closed-view head mounted displays (HMDs). Figure 1.1 illustrates both types of see-through mechanism with overview diagrams.

In addition to the two original approaches to augmented reality, namely optical see-through and video see-through HMDs, several other methods were proposed in recent years. These include systems like the Virtual Showcase, which combines several semitransparent mirrors in a local multi-user setup [55], and projector-based approaches [56].



(a) Optical see-through HMD (b) Video see-through HMD

Figure 1.1: Comparison of optical see-through and video see-through head-mounted displays for augmented reality (adapted from [43]).

This thesis focuses on video see-through augmented reality. The medical augmented reality system described in Chapter 2 uses a video see-through approach for image combination. Video see-through augmented reality is also required if stylization filters are to be applied to the camera image, a principle used by the techniques described in Chapter 3 and 4.

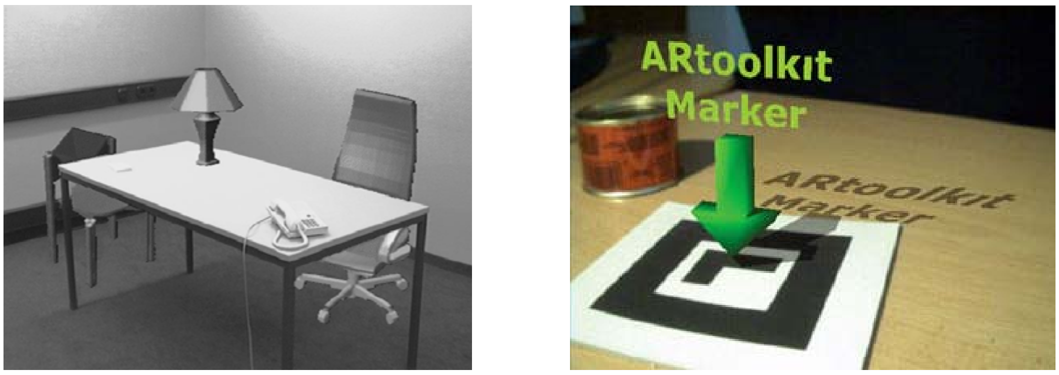
## 1.2 Rendering Methods for Augmented Reality

In the last years, researchers have explored many different potential applications of augmented reality. One of the first and most intensively examined scenarios is the support of medical diagnostics and treatment [86, 150, 186]. In Chapter 2 of this thesis, a novel AR system for medicine is presented, which uses commercially available surgical equipment for accomplishing a useful augmentation. Another promising direction of research is the application of augmented reality in an industrial context. The use of AR technology has been demonstrated for assisting design and development, production, training, and maintenance in industry-related projects [166]. Moreover, various other scenarios for augmented reality have also been proposed. These include applications in entertainment [193] and cultural heritage [200], among many others.

With regard to the underlying technical challenges of AR, many research efforts focus on the topics of calibration and registration, hardware setups, and system design aspects like authoring. For instance, vision-based tracking algorithms for augmented reality have continually been improved during the last years [119]. Many different and advanced techniques for AR displays and image combination have been explored [54]. The problems of designing complete augmented reality systems and authoring content for AR applications have also been intensively studied [107, 132].

Compared to these directions of research, only a relatively limited amount of previous work exists on the topic of novel methods for rendering in augmented reality. The majority of existing AR systems uses standard computer graphics algorithms for displaying virtual objects in an augmented environment. Widespread software libraries like OpenGL [181] and high level scene graphs based on them (e.g., OpenSG PLUS [184]) are often utilized for real-time rendering in AR. Such proven technologies are easy to apply in an augmented reality pipeline and are inherently fast enough for generating interactive AR displays. The generated output, however, tends to look artificial and contains typical computer graphics artifacts like aliasing caused by the rasterization process and simplified lighting models.

Some researchers have tackled the problem of specialized rendering for augmented reality. One of the first approaches for improving the visual quality of displayed virtual objects was occlusion handling. Occlusion handling tries to resolve visual ambiguities caused by the fact that graphical objects are always overlaid over the background image in standard AR. The graphical objects, therefore, always occlude the real world, regardless of the actual spatial relationships in the observed scene. Geometry-based [64] and image-based [52] algorithms were proposed for handling occlusion in augmented reality. Figure 1.2a shows an example of occlusion handling in AR. Another method for improving the visual realism of rendered augmented reality images is the addition of virtual shadows to the scene [104]. An augmented image containing a virtual shadow based on a known geometrical model is shown in Figure 1.2b. Some research has also been done on analyzing the illumination conditions in the real environment [41]. This way, graphical objects can be rendered with a more realistic lighting model.



(a) Static occlusion handling with phantom models. In this image, the chairs and lamp are virtual models. The virtual chairs are correctly occluded by the real table (from [64]).

(b) Virtual shadows in augmented reality. The graphical objects (arrow and text) cast a shadow on the real table (from [104]).

Figure 1.2: Improved realism by means of advanced rendering methods in augmented reality.

As its core contribution, this thesis seeks to extend the existing work on novel methods of rendering for augmented reality. In the medical augmented reality system described in Chapter 2, a new occlusion handling scheme is used in order to achieve a more easily understandable display. With this method, the occlusion of graphical objects by the anatomy of the patient can correctly be displayed, leading to a better representation of depth relationships in the scene.

In Chapter 3, a different method of generating AR images is introduced, which applies stylization methods to the entire augmented reality scene. By using the same type of artistic or illustrative stylization for both the real and the virtual elements in an augmented video stream, they become less distinguishable. Therefore, a novel AR experience is created. This approach was proposed for the first time in the context of this thesis. Subsequently, a novel algorithm for the illustrative visualization of volumes and polygonal datasets is described in Chapter 4, which is then also applied to augmented video streams as the basis for an illustrative AR system.

## 1.3 Overview and Contributions

The results presented in this thesis were achieved in two main steps. In the first phase, a new medical augmented reality system, *ARGUS*, was designed and realized (see Chapter 2). The software component of this medical AR system was implemented as a flexible framework for augmented reality. *ARGUS* constitutes an important milestone of project VIRTUE, which is the context in which this thesis was completed. The aim of project VIRTUE is the combination of surgical navigation, modern modes of visualization, and endoscopy in real-life medical applications [3, 49, 50]. This project is supported by the German Research Foundation (DFG) in the focus program on ”Medical Navigation and Robotics” (SPP 1124). In addition to the *ARGUS* framework, a common software core for advanced, platform-independent rendering and visualization was also realized. This software core has the same name as the project for which it was created, *Virtue*.

These software components, *ARGUS* and *Virtue*, serve as the basis for the topics which dominated the second phase of research presented in this thesis. Both the novel stylization approach for augmented reality (see Chapter 3) and the new illustrative visualization algorithm (see Chapter 4) were implemented using this software framework.

In addition to the main topics mentioned so far, several smaller projects were also carried out. The following list enumerates the contributions described in this thesis, as well as the associated improvements upon the state-of-the-art (at the time of first publication):

- The medical augmented reality system *ARGUS* was designed and implemented [22]. Unlike the vast majority of previously described AR setups for medicine, this system uses commercially available surgical navigation equipment for tracking and patient data import. This eliminates the need for specialized hardware components.

- As an application case for the newly developed medical visualization and augmented reality methods, the task of operation planning for maxillofacial surgery was examined. A new software tool takes symmetry considerations into account for the preparation of a so-called orbital reconstruction intervention [21].

- An advanced hybrid tracking system for medical augmented reality was developed. This approach combines tracking information from an infrared camera system with computer vision methods in order to improve the camera pose estimation in the *ARGUS* framework [20].

- Based on the *ARGUS* system, a new method for real-time occlusion handling in medical augmented reality was realized, which utilizes the available patient data [10].

- Novel approaches to user interaction in medical AR were explored. A new three-dimensional user input system provides support for basic gestures and a configurable menu system [13]. The user interaction is performed with untethered interaction tools tracked by a surgical navigation system, requiring no additional hardware.

- A new method for generating augmented video streams in video see-through AR was proposed. This novel approach, *stylized augmented reality*, applies similar types of artistic or illustrative stylization to the real background image and the virtual objects [15]. This way, real and virtual scene elements are less distinguishable from each other, which can lead to an improved immersion in the augmented environment. The work presented in this thesis describes the concept of stylized augmented reality for the very first time. As the first implementation of this principle, an algorithm for the cartoon-like stylization of AR images was designed and implemented.

- As an extension of the stylized augmented reality concept, a technique for rendering augmented video frames in a brush stroke style was devised [11].

- In order to achieve a better implementation of the principle of stylized AR, an improved algorithm for the cartoon-like stylization of augmented reality images was developed [14]. This new method is a postprocessing filter, which is executed on the graphics processing unit (GPU). It delivers significantly higher frame rates and a better visual quality compared to the original approach.

- A psychophysical study on the effectiveness of stylized augmented reality was performed. In the study, participants were asked to judge whether objects shown in AR video sequences are real or virtual. The results of the study confirm that the application of stylization methods to augmented video streams significantly reduces the discernability of virtual objects [16].

- A novel method for the illustrative rendering of iso-surfaces generated from volume datasets was proposed [12]. This new real-time illustrative visualization algorithm automatically displays the inner structures of objects in indirect volume rendering. No preprocessing of the data is necessary. The method uses an optimized combination of object space and image space processing passes executed on the graphics processing unit (GPU). The algorithm can also be used for displaying general polygonal models. This technique was the first to make the illustrative display of inner structures of iso-surfaces on the GPU possible in real-time without preprocessing.

- The new illustrative rendering algorithm was applied to augmented video streams, creating another novel way of rendering AR images. In illustrative AR, both the camera image and the virtual objects are displayed in a black-and-white technical illustration style.

The remainder of this thesis is structured as follows. In Chapter 2, the medical augmented reality system *ARGUS* and its extensions, i.e., the hybrid tracking, user interaction, and occlusion handling techniques, are described. Chapter 3 introduces the concept of stylized augmented reality and the newly developed stylization algorithms. The new illustrative visualization method and its application to illustrative AR are described in Chapter 4. Finally, Chapter 5 concludes this thesis with a summary and an outlook on possible future developments.

CHAPTER 2

# Medical Augmented Reality

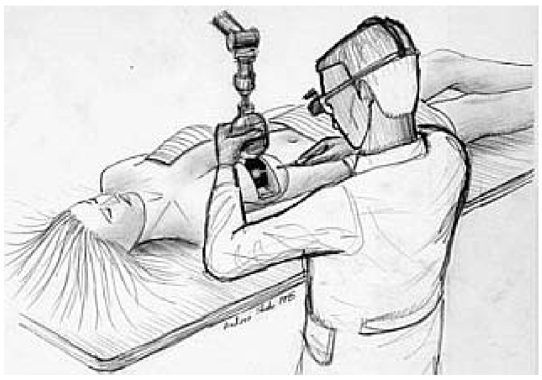
## 2.1 Augmented Reality for Medical Diagnostics and Therapy

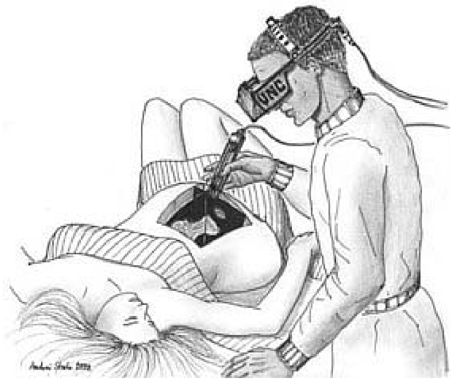
Medical diagnostics as well as the planning, preparation, and support of medical treatment have always been among the most important applications of augmented reality. The aim of medical AR is the overlay of useful additional graphical information directly over the patient. Such additional information can include the visualization of preoperatively acquired patient data (e.g., three-dimensional MRI or CT scans), the display of operation plan elements created in the preparation phase, and information related to the equipment used in an intervention (e.g., the position of surgical tools).

The need for the visualization of additional information during surgical interventions has increased in recent years due to the more frequent use of minimally invasive techniques. In minimally invasive surgery, operations are performed with the help of small cameras (endoscopes) and specially designed instruments, which are inserted through small keyhole sized surgical incisions. This type of surgery offers many benefits for the patient, including less blood loss, minimal scarring, and shorter hospital stays [71]. One challenge during minimally invasive interventions is the fact that the surgeon's view is limited to the endoscopic image, which only shows a small region of the patient's anatomy. It can therefore be difficult to determine the current position and orientation of the endoscope and surgical instruments relative to anatomical structures. A widespread approach for the visualization of anatomical structures on a standard screen during the planning phase of minimally invasive surgery is the so-called virtual endoscopy [48, 115].

The use of augmented reality for supporting minimally invasive procedures or even replacing conventional endoscopy has been studied for a long time. An early example is the ultrasound visualization research performed at the UNC Chapel Hill since the early 1990s [45].

Figure 2.1 shows concept sketches of two possible applications of augmented reality for ultrasound visualization. In these sketches, live ultrasound images are displayed as graphical overlays over the patient to the surgeon, who is wearing a head-mounted display.


(a) Concept sketch: Ultrasound-guided breast biopsy in AR



(b) Concept sketch: Fetal ultrasound examination in AR

Figure 2.1: Concept sketches of augmented reality for the visualization of live ultrasound images (from [196]). In the sketches, video see-through augmented reality is used for overlaying additional information over the patient.

In recent years, a large number of experimental augmented reality systems for medicine were created. In addition to the technical challenges described in Chapter 1, several application-specific problems also have to be solved when designing a medical AR setup:

1. **Patient registration**. Since any useful overlay of medical information is always relative to the patient's anatomy, the exact location and orientation of the patient have to be known. This problem is not trivial and can usually only be solved reliably for rigid parts of the anatomy, e.g., the skull of the patient. It also typically requires that the patient remains completely immobile during the intervention. In many medicial augmented reality systems, patient registration is performed by means of specialized hardware components like infrared tracking cameras or application-specific algorithms (e.g., image-based registration methods).

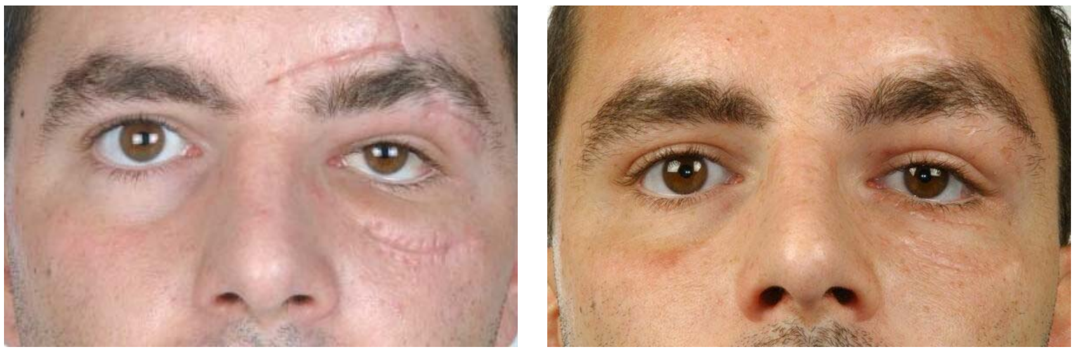
2. **Tracking of surgical instruments**. In particular during minimally invasive procedures, it is necessary for the surgeon to know the exact position and orientation of surgical instruments. This is especially important for the endoscope, but also for other tools, which are used for performing the actual operation. In many experimental medical AR systems, surgical instruments are also tracked with dedicated specialized hardware.

3. **Import of medical data**. A minor technical, but often tedious aspect of implementing a medical augmented reality application is the import of medical datasets. For most scenarios, three-dimensional volume datasets containing the relevant parts of the patient's anatomy are required. Such image data in medicine usually originates from 3D scanning devices like CT (computed tomography) or MRI (magnetic resonance imaging). This data is generally stored according to the DICOM standard (Digital Imaging and Communications in Medicine), which is notoriously complex [158]. Moreover, modifications and derivations from the standard are often encountered.

In the remainder of this chapter, first an application case for the medical AR and visualization techniques described in this thesis is discussed (Section 2.1.1). In Section 2.2, an overview of previous work in medical augmented reality is given. Section 2.3 describes the novel medical AR system *ARGUS*. In Section 2.4, a hybrid tracking algorithm developed for the *ARGUS* system is discussed. A description of the newly designed user interaction system contained in the *ARGUS* framework is given in Section 2.5. The new occlusion handling method for medical augmented reality is discussed in Section 2.6. Finally, Section 2.7 concludes this chapter with a summary.

### 2.1.1 Application Case: Intervention Planning for Orbital Reconstruction Surgery

In the context of this thesis, one concrete type of surgical intervention was explored as a potential application case for the described augmented reality and medical visualization methods. In the practice of maxillofacial surgery (upper jaw and face surgery), one major challenge is the correct reconstruction of malformed or damaged bone structures of the face. Among those, a rather frequent task is the remodeling of an injured orbit (eye socket). Injuries of the orbit occur frequently as a result of various types of accidents. Damage to the orbital cavity can also result from the preceding removal of a tumor in this area. A malformed orbit causes one eye to be displaced from a position that is symmetrical to the eye in the healthy half of the face. This can make the face look unesthetic and can also impair the function of the eyes, resulting in problems with the patient's vision. Figure 2.2a shows an example of a patient with one displaced orbit. In order to reconstruct an injured orbit, bone pieces are removed from the skull of the patient. These are then placed in the damaged orbit so that a more symmetrical placement of the affected eye is achieved. The result of such an intervention can be seen in Figure 2.2b.



(a) Before the intervention (b) After the intervention

Figure 2.2: A case of a patient with an injured orbit (source: Jürgen Hoffmann, Department of Oral and Maxillofacial Surgery, University Hospital Tübingen).

The main difficulty of this type of operation is the necessity to plan the repositioning of bone fragments so that symmetry is achieved. This usually has to be done manually using 2D slice images from scans of the patient's bone structures. In order to provide a better support for this task, a new experimental operation planning software for orbital reconstruction was developed [21]. This software tool was implemented by Melissa Mekić as part of her *Diplomarbeit* (Master's thesis) [146]. The planning tool, named *Schneewittchen*, provides a specialized user interface for defining a three-dimensional symmetry plane for the bone structures in the patient's skull. The definition of the symmetry plane is shown in Figure 2.3.

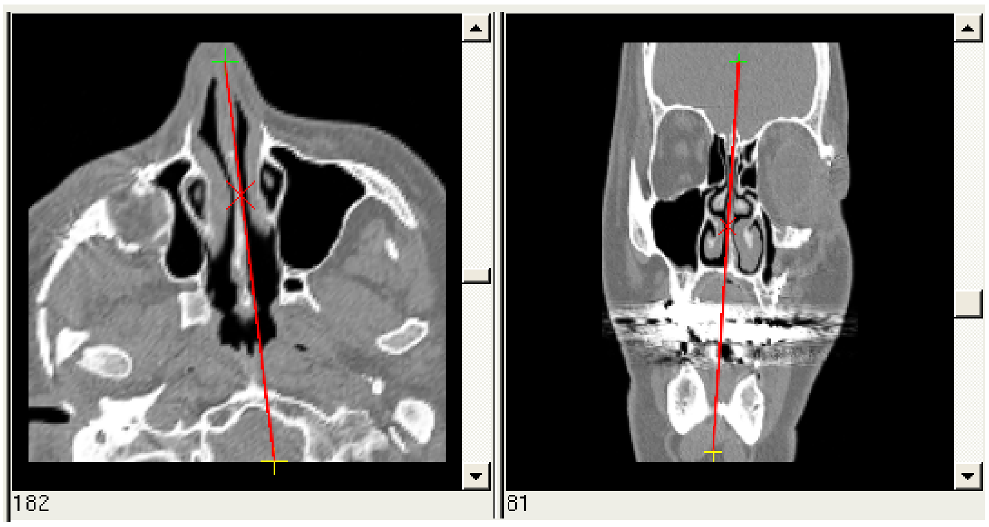
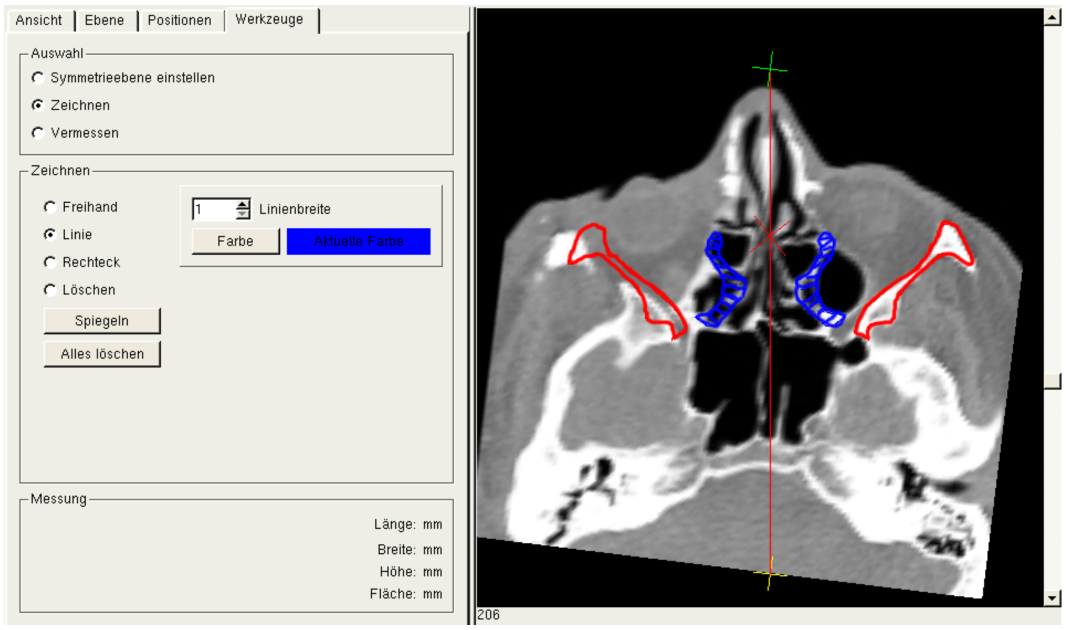


Figure 2.3: User interface for specifying the symmetry plane in the *Schneewittchen* software. This process is based on the definition of two-dimensional, partial "axes of symmetry" in orthogonal image slices.

After the symmetry plane has been specified, the user can highlight important anatomical structures. These drawings are automatically mirrored in 3D and overlaid over the associated slice images of the patient scan. This makes it possible to compare bone structures in the damaged and healthy portions of the patient's skull. Figure 2.4 illustrates this phase of the intervention planning process. The generated drawings and mirrored elements can then be exported in various ways for further processing in other planning software or for use in intraoperative navigation.

One technical challenge of the implementation of a planning tool for orbital reconstruction is the necessity for rotating the patient volume dataset. Since the symmetry plane is truly three-dimensional, the mirrored counterparts of highlighted anatomical structures are usually located in a different image slice of the original volume. They can even be distributed over several slices. In order to facilitate an intuitive interaction with displayed 2D slices, the volume dataset has to be rotated according to the symmetry plane parameters. This step requires a time-consuming resampling of the entire dataset. Therefore, a novel algorithm for applying rigid transformations to volume datasets was developed [18, 19]. It was shown that this new method can significantly speed up the transformation process.

This research on intervention planning for orbital reconstruction surgery was performed in close cooperation with the Department of Oral and Maxillofacial Surgery of the University Hospital Tübingen. The results of this work have influenced some aspects of the actual clinical practice of orbital reconstruction [28, 30, 31, 33, 34, 35]. Based on the experience with the *Schneewittchen* software, a planning tool for maxillofacial surgery using a polygonal representation of bone structures was later also developed [172].



Auswahl
Symmetrieebene einstellen [✓ Zeichnen] Vermessen

Zeichnen
Freihand [✓ Linie] Rechteck Löschen
Linienbreite: 1

Messung
Länge: mm
Breite: mm
Höhe: mm
Fläche: mm

Figure 2.4: Important anatomical structures are highlighted and mirrored at the symmetry plane.

## 2.2 Related Work

As mentioned above, the use of augmented reality for medical diagnostics and therapy has been in the focus of active research for many years. A method for overlaying live ultrasound images over the patient was presented by Bajura et al. in 1992 [45]. In 1996, State et al. described an experimental AR system for ultrasound-guided needle biopsies [186], as well as a specialized hybrid tracking algorithm for this scenario [185]. This system offers stereoscopic rendering and accurate tracking of both the user and the ultrasound probe, but relies on a number of proprietary technologies. These include a magnetic tracker for the user's head and a mechanical arm for probe registration.

In recent years, experimental AR systems were created for the support of various medical application scenarios. Navab et al. combined a specialized mobile X-ray system, a so-called C-arm, with a CCD camera in order to overlay acquired volume data over the optical image [150]. A head-mounted augmented reality operating microscope, the Varioscope AR, was presented by Birkfellner et al. [57]. It was later developed further and improved [85, 86]. The Varioscope AR is an optical see-through system which is based on a commercially available operating microscope with specially modified optical components. In order to make a spatially correct overlay of information possible, the operating microscope is tracked with dedicated optical tracking cameras. Sauer et al. have described a video see-through augmented reality system for the visualization of ultrasound images [171]. In this system, a dedicated infrared camera, which is attached to the head-mounted display worn by the user, is used for tracking. The AR system prototype of Splechtna et al. [183] consists of an optical tracking system comprising two cameras and a head-mounted display. It is capable of augmenting the surgeon's view with preoperative volume data or live ultrasound images. Their target application is liver surgery, during which a visualization of the vessel tree is especially useful. The Medarpa project, which uses a special translucent display device mounted on a swivel arm, was described by Schwald et al. [178, 179]. Here, tracking is performed using a hybrid scheme based on active infrared and electromagnetic components. Scheuering et al. have presented a two-stage approach to registration for augmented reality [173, 174]. In addition to rigid tracking using a hybrid optical-magnetic system, a non-linear deformation of patient data is performed based on image processing methods. Minimally invasive liver interventions are the target application for this system. Another augmented reality setup for supporting livery surgery was described by Bornik et al. [59, 60]. In this system, stereoscopic optical see-through head-mounted displays are used for displaying information from a liver surgery planning system. Dedicated optical cameras are used for tracking. A semi-transparent mirror display mounted directly over the patient is the basis for reality augmentation in the ARSys-Tricorder project described by Goebbels [96]. It provides stereoscopic rendering, with infrared cameras being used for tracking the surgeon's head and components of the AR system. Bockholt et al. describe a concept for overlaying preoperatively acquired patient scans and operation plan information during an intervention [58].

Among the latest developments in medical augmented reality is the method described by Feuerstein et al. [84] for supporting optimal port placement in robotically assisted heart surgery. A specially designed planing process is used for this task [51], and the resulting planning information can be overlaid over images delivered by an endoscope. Olbrich et al. propose a partial periodic augmentation of endoscopic images during minimally invasive liver surgery [157]. They limit the display of additional information to certain phases of the respiration cycle in order to avoid overlay errors caused by liver motion.

### 2.2.1 Drawbacks of Previous Systems for Medical Augmented Reality

The main drawback of many existing methods for realizing augmented reality in medicine is their reliance on specific hardware. Such devices, like dedicated magnetic or optical tracking systems and specialized displays, often are expensive and can require tedious setup procedures. Many of them have originally been designed for applications in industry or virtual reality and are not optimally suitable for medical scenarios. In particular stereo shutter glasses and head mounted displays are usually considered to be too bulky, and they may deteriorate the user's perception of the surroundings. Moreover, connecting cables used for data transmission or power supply can limit the user's range of motion. They are thus not well accepted for medical applications.

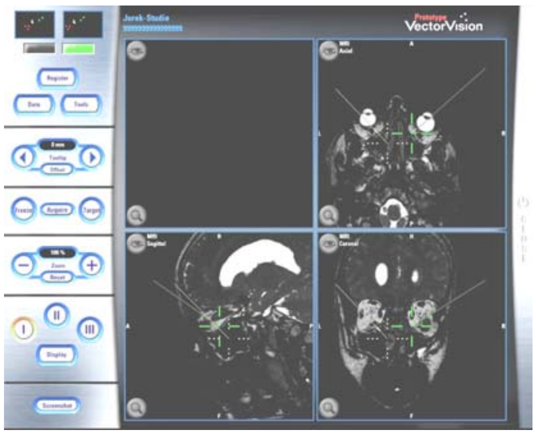
Magnetic tracking devices designed for VR scenarios, which are used in many existing medical AR systems, may prevent the transition from an experimental state into the clinical practice. Such magnetic trackers can be disturbed by metallic objects like surgical instruments, and they can also possibly interfere with other sensitive equipment in the operating room. Although the use of specialized magnetic trackers in medical applications is currently being investigated (e.g., see [87]), they have only recently started to become available as commercial products. Moreover, even specialized magnetic trackers for intraoperative navigation can still be disturbed by other intraoperative equipment like C-Arm devices for X-Ray scans. The very limited space available in the operating room also prevents the utilisation of some types of devices. Furthermore, practical problems like working in a sterile environment and certification for medical settings usually have not been solved for specialized VR and AR tracking and display equipment.

Another difficulty of medical AR is the fact that a number of problems related to registration, tracking and calibration need to be solved. These may require a lot of effort for the design, implementation and test of the necessary algorithms and software components.

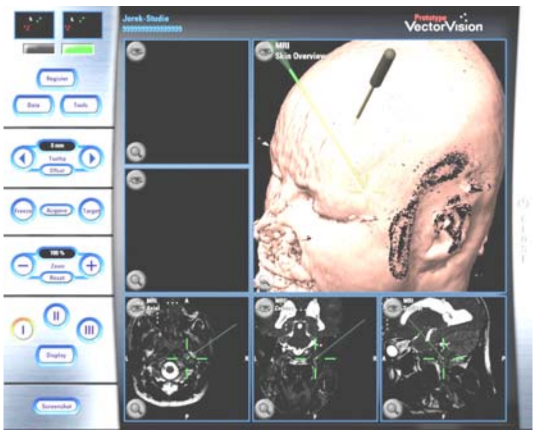
## 2.3 Augmented Reality based on Image Guided Surgery

In the context of this thesis, a novel medical augmented reality system was developed. This new system, *ARGUS*, uses commercially available medical equipment for solving many problems of medical AR. This way, many of the difficulties mentioned in Section 2.2.1 can be avoided.

*ARGUS* is based on a so-called image guided surgery device (the name of the system is an acronym for Augmented Reality based on image GUided Surgery). Image guided surgery (IGS) is a technology which has become a widespread tool for modern types of surgery such as minimally invasive interventions. The basic concept of image guided surgery is that the surgical procedure is facilitated by a real-time correlation of the operative field to a monitor, which shows the precise location of selected surgical instruments to the surrounding structures [149]. This concept is illustrated in Figure 2.5, which shows two screenshots from an IGS system. Here, a patient dataset is depicted, which was acquired preoperatively with an MRI scan. The patient's anatomy can be displayed as image slices from the volume dataset (see Fig. 2.5a) or as a three-dimensional model (see Fig. 2.5b). In addition to the patient data, the position and orientation of surgical instruments can be seen. The surgical instruments are tracked by the IGS system during the operation. Since a highly accurate patient registration is performed before the intervention, it is possible to display the exact location of the instruments relative to the patient's anatomy. Such a real-time visualization of the spatial relationship between tracked surgical tools and anatomical structures during a surgical procedure is also called intraoperative navigation.



(a) IGS display showing orthogonal image slices



(b) IGS display showing a three-dimensional model of the patient anatomy

Figure 2.5: Real-time visualization of patient data and tracked surgical tools in an IGS system. These images are screenshots from the VectorVision® Cranial software developed by the BrainLAB company (Heimstetten, Germany). In the top right portion of Fig. 2.5b, the tracked surgical instruments are displayed as three-dimensional models.

Image guided surgery has been used as the basis for advanced modes of intraoperative visualization. Weber has described a mobile navigated image viewer [204]. This viewer consists of a small mobile TFT display, which is tracked so that its position and orientation relative to the patient are known. The display can be moved freely in the proximity of the patient. The viewer shows 2D image data based on a preoperatively acquired patient scan. These images, which represent a section of the patient anatomy parallel to the TFT display, are generated in real-time.

### 2.3.1 VectorVision and the VectorVision Link library

A VectorVision® image guided surgery device is the basis for the *ARGUS* augmented reality system. The VectorVision platform is manufactured by the BrainLAB company (Heimstetten, Germany) [63]. Figure 2.6 shows a VectorVision compact IGS system, which is the type that is used for developing and extending *ARGUS*. The image guided surgery device consists of a computer system running the IGS software, a touchscreen LCD display, and an infrared camera system. The infrared cameras track surgical instruments in real-time with a very high accuracy.

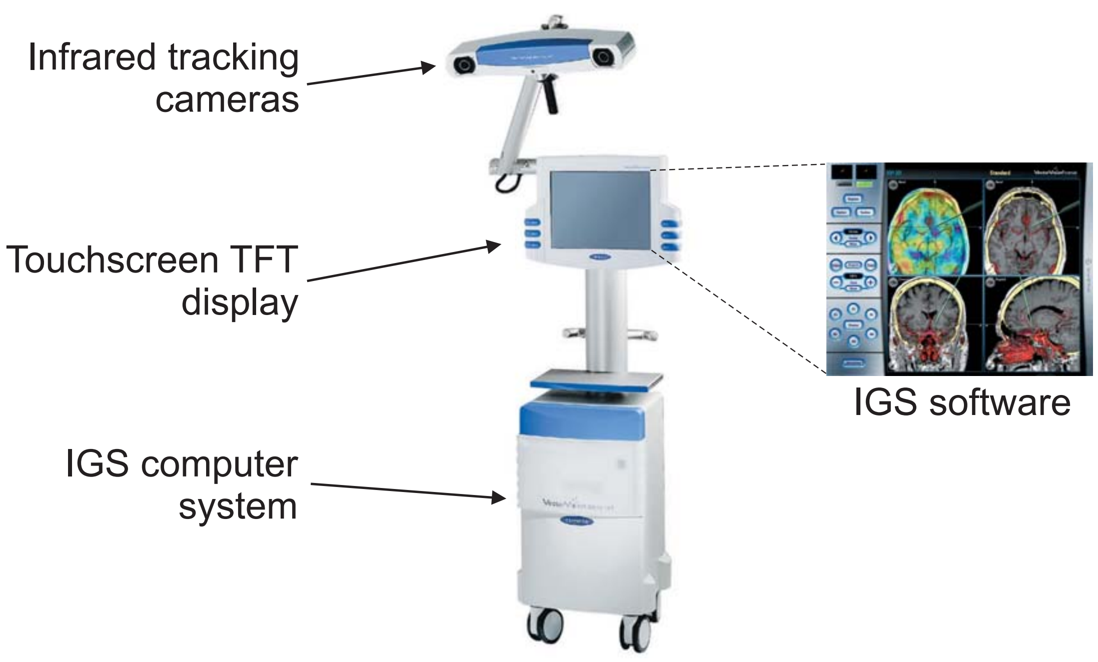


Figure 2.6: VectorVision® compact image guided surgery system manufactured by the BrainLAB company (image source: company website [62]).

An infrared source integrated into each camera emits infrared waves. Special marker spheres that are rigidly attached to every object to be tracked by the IGS system reflect the infrared light back to the cameras. The position of each marker sphere is calculated via triangulation, which allows the touchscreen to display the 3D model of the surgical tool at the correct location in the 3D dataset. The triangulation step is depicted in Figure 2.7.

Disregarding the fact that soft tissue can slightly modify shape and position over time, in particular between the image acquisition and the surgical intervention, the technically possible accuracy achievable today is in the order of magnitude of 1 mm [114].

It is one particular feature of the VectorVision platform that it provides a network interface for accessing internal IGS data. This interface, VectorVision Link, consists of an easy-to-use TCP/IP-based protocol [151]. A software library is available for accessing VectorVision Link

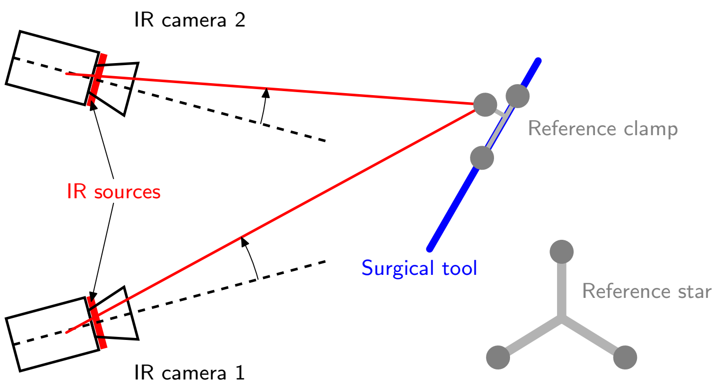
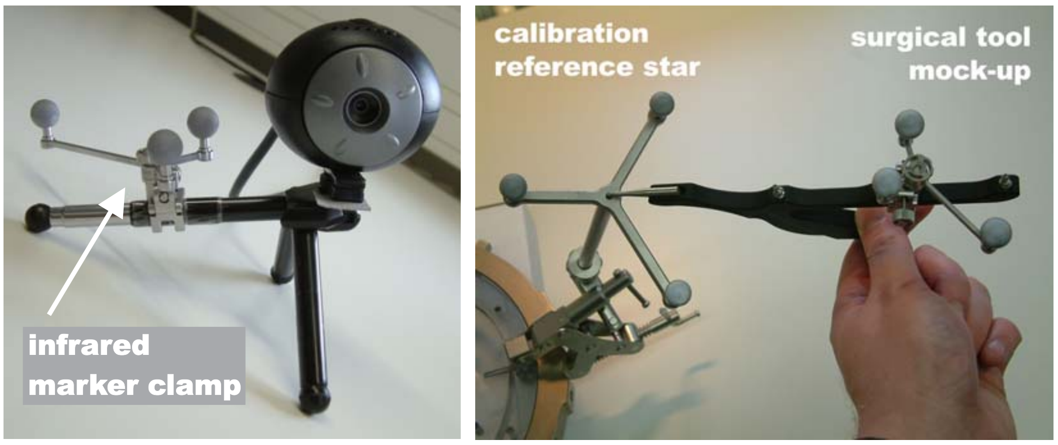


Figure 2.7: Infrared tracking of surgical instruments.

(VVL) from different platforms (Linux and Windows operating systems) and programming languages (e.g., C++ and Java). Many different types of data can be exchanged through the VVL interface between the IGS system and an external computer. These include patient data and the position and orientation of tracked surgical instruments. This network access to internal data of the image guided surgery device is the basis for reality augmentation in *ARGUS*. The research presented in this thesis was among the very first to use the VVL protocol for advanced modes of medical visualization [8, 24]. *ARGUS* is the first medical augmented reality system which uses this method of transferring IGS data over the network for camera tracking.

### 2.3.2 Camera Tracking in the ARGUS System

*ARGUS* is designed as a video see-through system. An off-the-shelf webcam is used for the acquisition of the video images. As one central aspect of the design of the new AR system, a method for tracking the webcam using the given equipment was developed in the context of this thesis [22, 23, 25]. The VectorVision device is capable of tracking surgical instruments using its built-in infrared cameras. This capability is harnessed by attaching an instrument adapter clamp with infrared marker spheres to the webcam. The setup is shown in Figure 2.8a.



(a) Webcam with attached passive infrared markers

(b) Calibration of surgical tool in the IGS system

Figure 2.8: Webcam with infrared markers and calibration of surgical tool in the IGS system.

The position and orientation of the instrument clamp measured by the IGS system is received using the VectorVision Link interface. Although the latency depends on the system load of the VectorVision device and the current network traffic, real-time transfer of the data is almost always possible.

### Tracked Instrument Pose Data

The major difficulty of using tracked instrument data for tracking the webcam is the fact that the pose information delivered by the IGS system does not directly correspond to that of the camera. In order to utilize an instrument clamp, it has to be calibrated as if it was attached to an actual surgical instrument. This is done in the IGS software using a special procedure aimed at ensuring accuracy and correct placement of medical tools within the patient's coordinate system. Figure 2.8b shows how a mock-up of a surgical tool is calibrated with respect to a so-called reference star (seen at the left). The same instrument clamp used for the webcam in Figure 2.8a is attached to the instrument mock-up.

Since this calibration procedure requires the instrument to have a fixed physical tip, it is not possible to directly calibrate the webcam at the reference star. Thus the AR system can only obtain position and orientation information in relation to the instrument mock-up, as shown in Figure 2.9. In this figure, **pos** denotes the position of the tip of the instrument. The direction of the trajectory of the instrument is given as vector **dir**, with **norm** being a plane normal perpendicular to it. By combining the information contained in **pos**, **dir** and **norm**, the complete six degrees of freedom (6-DOF) pose information can be obtained. It is then necessary to transform this pose, which is relative to the instrument used for the initial reference star calibration, into the camera pose required for the AR image composition. An additional one-time calibration step for computing this transformation was devised.

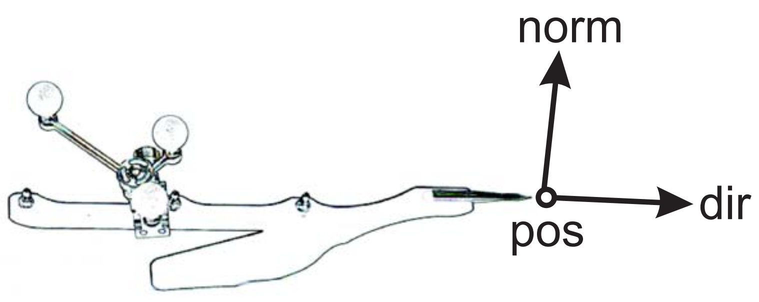


Figure 2.9: Six degrees of freedom (6-DOF) instrument data delivered by the image guided surgery system.

### One-time Calibration

A matrix describing the transformation from the tracked instrument to the camera is computed. In order to be able to perform this computation, a common reference coordinate system is established at a given point of time. This reference coordinate system contains both the tracked instrument and the webcam in absolute positions and orientations. A transformation between the two can then be easily derived. For this one-time calibration, conventional marker tracking based on the ARToolKit by Kato et al. is utilized [123].

An ARToolKit marker is placed in or near the trackable volume of the VectorVision infrared cameras. To make the measurements and the calibration stable and repeatable, a physically fixed relation between the reference star and the marker is ensured. The user then defines the positions of the marker corners in the coordinate system of the infrared camera. A standard pre-calibrated pointer tool is used for indicating the spatial positions of the corners. The pointer tool is automatically recognized by the image guided surgery system, and the position of its tip can be retrieved through the VectorVision Link interface. Figure 2.10 shows the definition of two marker corners.

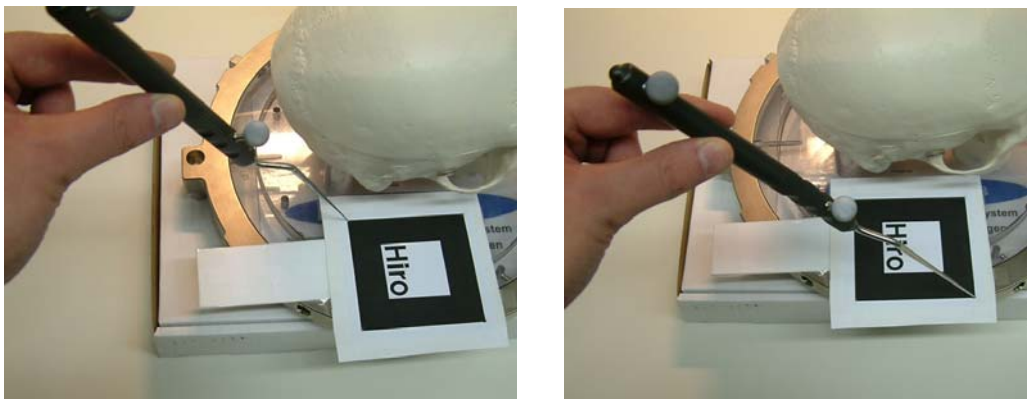


Figure 2.10: Two marker corners are indicated by the user with the pointer tool.

The user is prompted to define the corners in clockwise order as shown in Figure 2.11. Therefore, the transformation matrix from the VectorVision coordinate system to the coordinate system of the marker can easily be constructed. The marker coordinate system is the same one as used in the ARToolKit: Vectors **u** and **v** in Figure 2.11 correspond to the x and y axes, respectively. The z axis points upward towards the viewer, creating a right-handed coordinate system.

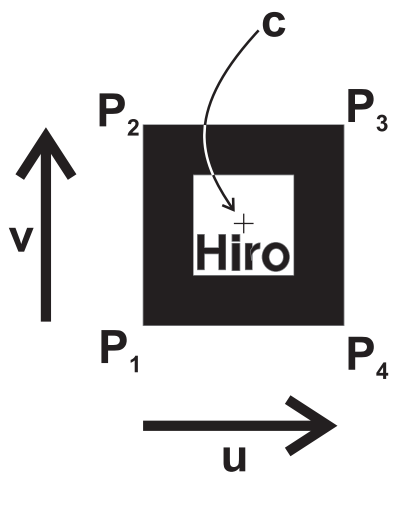


Figure 2.11: Marker corners are defined in a clockwise order from $P_1$ to $P_4$. The marker shown here is an ARToolKit marker [123].

The transformation matrix from the VectorVision to the marker coordinate system, denoted here as $A_{vvctomarker}$, is computed according to Equation 2.1. The origin of the marker coordinate system is the geometric centroid of the four defined corner points. Note that both

**u** and **v** are normalized to a length of one. Other considerations concerning the scaling of the coordinate systems are not necessary, since both VectorVision and the ARToolKit define a coordinate unit as equivalent to one millimeter.

$$
\begin{gathered}
\mathbf{u} := \frac{P_4 - P_1}{\|P_4 - P_1\|} \qquad \mathbf{v} := \frac{P_2 - P_1}{\|P_2 - P_1\|} \\
\mathbf{c} := \frac{P_1 + P_2 + P_3 + P_4}{4} \\
A_{vvctomarker} := \left( \begin{array}{c|c|c|c} \mathbf{u} & \mathbf{v} & \mathbf{u} \times \mathbf{v} & \mathbf{c} \end{array} \right)^{-1}
\end{gathered}
\tag{2.1}
$$

In the next step, the actual one-time calibration procedure is triggered by the user. At one point of time, the webcam has to be positioned so that the optical marker can be recognized by the ARToolKit, while the infrared markers are visible to the VectorVision cameras simultaneously. The computation of the necessary transformation matrices is triggered with a single keystroke.

The matrix for the transformation from the VectorVision coordinate system to the coordinate system of the tracked instrument marker clamp is then calculated. When the calibration is performed, the current tracked instrument data, as shown in Figure 2.9, is recorded. This pose information directly leads to the construction of matrix $A_{vvctoinsm}$ (see Equation 2.2), which describes a right-handed coordinate system with the (imaginary) instrument tip at its origin.

$$
A_{vvctoinsm} := \left( \begin{array}{c|c|c|c} \mathbf{dir} & \mathbf{norm} & \mathbf{dir} \times \mathbf{norm} & \mathbf{pos} \end{array} \right)^{-1}
\tag{2.2}
$$

In the most recent versions of the VectorVision Link interface, the complete transformation matrix for an instrument clamp can be downloaded in addition to the **pos**, **dir** and **norm** vectors. For these versions of VectorVision Link, the *ARGUS* software simply uses the downloaded transformation data as the $A_{vvctoinsm}$ matrix.

Finally, the marker transformation matrix is retrieved from the ARToolKit. It is then inverted, yielding the transformation $A_{camtomarker}$ from the coordinate system of the webcam to the marker coordinate system. Using the ARToolKit marker as the common point of reference, it is now possible to compute the transformation from the instrument clamp to the webcam coordinate system. Figure 2.12 illustrates the relations between the transformation matrices involved in the calibration procedure.

$$
A_{insmtocam} = A_{camtomarker}^{-1} \cdot A_{vvctomarker} \cdot A_{vvctoinsm}^{-1}
\tag{2.3}
$$

Equation 2.3 shows how the result of the calibration, matrix $A_{insmtocam}$, is computed. Note that this transformation only needs to be computed once. As long as the physical relation between the instrument marker clamp and the webcam remains unaltered, the matrix remains valid. A new calibration is only required when the instrument clamp is moved relative to the webcam.

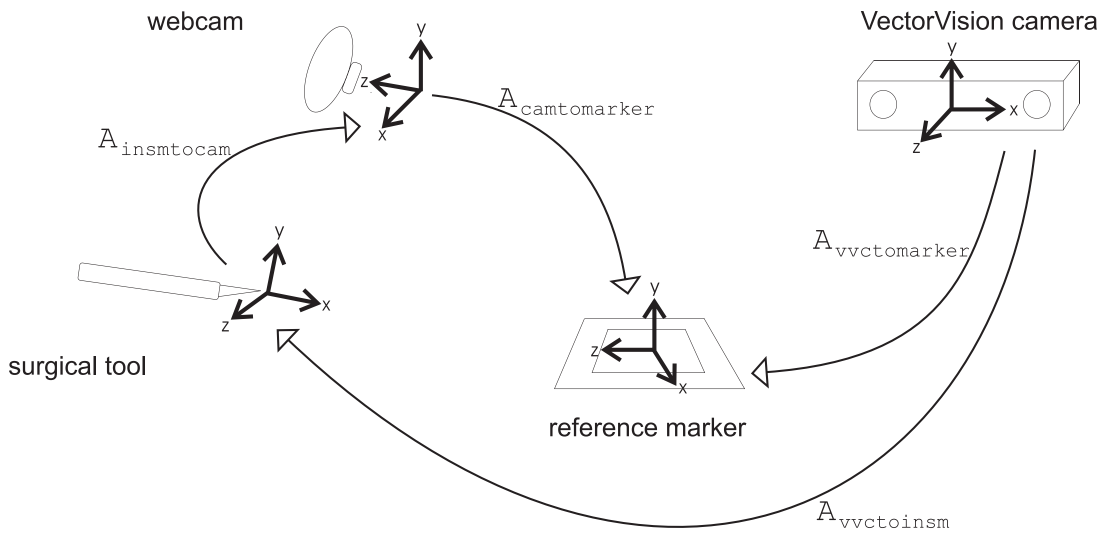


Figure 2.12: Overview of the transformation matrices involved in the calibration and tracking process.

### Operation of the Camera Tracking System

After the one-time calibration step, only the data from the VectorVision infrared camera is required for tracking the webcam. The vision-based marker tracking is not used for the operation of the augmented reality application itself. For every frame that is generated by the AR system, the current tracked tool data is retrieved using the VectorVision Link. An $A_{vvctoinsm}$ matrix is computed as described in Equation 2.2. The final transformation matrix used for OpenGL camera setup is the result of a simple multiplication as shown in Equation 2.4.

$$A_{vvctocam} = A_{insmtocam} \cdot A_{vvctoinsm} \tag{2.4}$$

## 2.3.3 Additional Support by the IGS System

In addition to the camera tracking described above, the image guided surgery system also provides support for several other tasks of medical AR. Surgical tools are tracked in the same way as the instrument clamp attached to the webcam (see Sec. 2.3.2). Several instruments can be tracked simultaneously, each delivering 5-DOF or 6-DOF pose information.

The VectorVision system offers several methods for patient registration. These include algorithms based on the matching of anatomical landmarks, artifical fiducials attached to the patient, or three-dimensional surface point clouds generated by a special handheld laser device.

A software pipeline for importing and preprocessing DICOM image datasets is provided by the IGS system. It is possible to define target points or trajectories for interventions. Moreover, volumes originating from different scanning modalities can be registered for use in a single session. All of this data can be exchanged with the augmented reality system using the VectorVision Link. The AR application can upload and download volume datasets, retrieve patient registration parameters and tracked instrument data, and read and add intervention planning information.

### 2.3.4 Overlay of Graphical Information with ARGUS

Several example cases for the overlay of graphical information were realized with the *ARGUS* framework. Figure 2.13 shows results from an early test application. Here, a graphical cube model is positioned at the origin of the coordinate system of the IGS infrared cameras. When moving the webcam in the trackable volume of the image guided surgery system, the cube model remains correctly positioned and oriented in the current camera image. The acquired camera image with the graphical overlay is displayed to the user in an interactive software application.

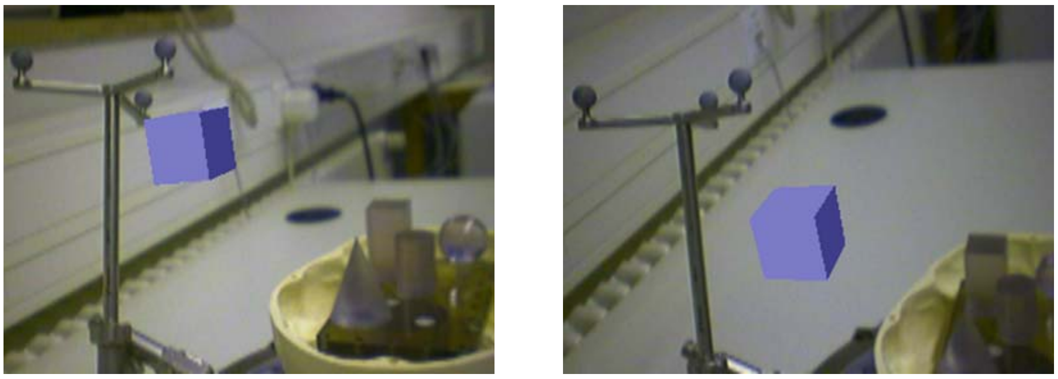


Figure 2.13: Early overlay experiments with *ARGUS*. The blue cube is a virtual object. It is displayed with a correct position and orientation relative to the reference star (seen at left) based on the current camera position.

Figure 2.14 shows an example application which demonstrates the use of augmented reality in a medical context. The red object in the images is the graphical model of a tumor generated from actual patient data. The model is manually positioned so that it is located inside a real plastic skull phantom[1]. This plastic skull is the test setting which was used during the development of *ARGUS*. Patient registration in the image guided surgery system was performed with an MRI scan of the phantom skull. As shown in Figure 2.14, here again the graphical object is displayed with correct spatial positioning and orientation thanks to AR camera tracking.

One of the main advantages of utilizing image guided surgery for camera tracking is the fact that patient registration is provided by the IGS device. For the *ARGUS* framework this means that all the tracking information, i.e., position and orientation data for the webcam and all tracked instruments, has a common coordinate system which is relative to the patient. This is possible because the IGS system contains several registration algorithms for the patient anatomy which rely on the automatic or manual identification of anatomical landmarks. Therefore, the transformation between the patient anatomy and a so-called reference star, which is also tracked by the infrared cameras, is known. The marker clamps attached to the webcam and surgical instruments are tracked relative to this reference star.

For the operation of the *ARGUS* system this means that a change in the position or orientation of the patient anatomy is handled automatically and transparently. The augmented reality application can define graphical objects in a global coordinate system which is relative to the patient. This is illustrated in Figure 2.15, in which the same virtual tumor model as in

[1]The graphical model was originally created from the scan of a lung tumor.

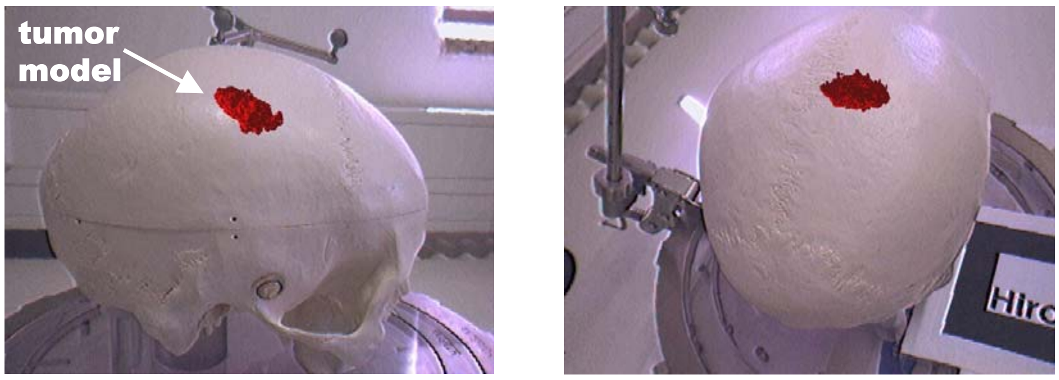


Figure 2.14: Overlay of a virtual tumor model over the camera image. The red object is a manually positioned virtual tumor model. It is overlaid over a plastic skull phantom (mock-up). Due to the camera tracking provided by *ARGUS*, the virtual tumor model remains correctly positioned and oriented inside the plastic skull.

Figure 2.14 is used. When recording this image sequence, the webcam remained at a fixed position, while the plastic skull was moved. Still, the graphical tumor model remains at the correct position relative to the phantom skull.

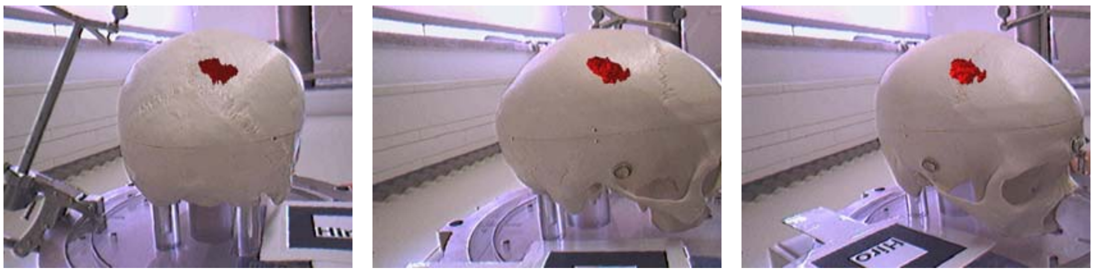


Figure 2.15: Automatic patient registration in *ARGUS*. These images were acquired with a fixed camera position, while the experimental setup (plastic phantom skull) was moved. Nonetheless, the virtual tumor model keeps its correct position and orientation relative to the plastic skull. This is possible due to the patient registration provided by the image guided surgery device.

In the *ARGUS* framework, augmented reality applications can also access the position and orientation information of tracked instruments. This is illustrated in Figure 2.16. Here, a graphical model representing a tracked tool is displayed in the AR environment. The instrument shown in this image is a so-called pointer tool, which is depicted as a yellow cylinder. The graphical object is attached to the actual tracked instrument. Therefore, they move synchronuously. While this application only demonstrates a relatively simplistic visualization of the position and orientation of tracked tools, this information can be very useful for more advanced types of visualization, e.g., for the display of instrument trajectories.

The *ARGUS* augmented reality system was developed and is currently operated using Firewire webcams as video see-through digital cameras. These Firewire cameras are capable of delivering a video stream with a resolution of 640x480 pixels at a frame rate of 30 Hz. The overall frame rate of the *ARGUS* system depends on several factors. In addition to the geometric complexity of the graphical objects in the augmented scene, which intrinsically has

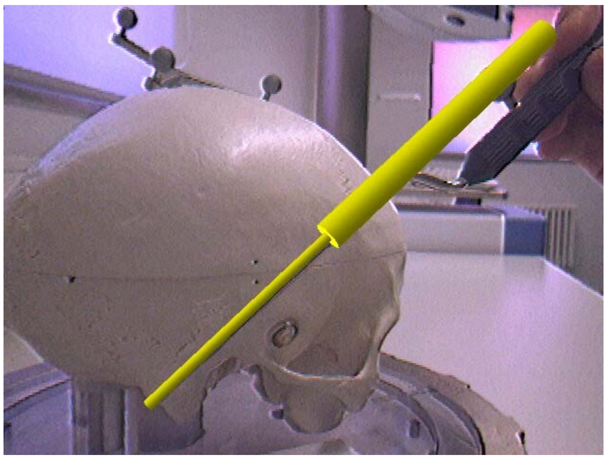


Figure 2.16: Overlay of the graphical model of a tracked instrument. The current position and orientation of the tracked tool is downloaded from the VectorVision device along with the camera tracking information. It is therefore possible to accurately draw the model of the tracked tool (yellow cylinders) over the camera image.

an impact on the rendering speed, the network latency of the VectorVision Link connection is a second important factor. This network latency depends on the speed of the local network infrastructure, but also on the current frame rate of the VectorVision software running on the IGS device.

The reason for this dependency of the frame rate of the *ARGUS* software on the performance of the IGS software is the fact that both are fully sychronuous systems. This is illustrated in Figure 2.17. At the beginning of each iteration of the *ARGUS* main rendering loop, the current tracking information is requested from the VectorVision device. These data are the basis for AR camera tracking, which is necessary for a spatially correct overlay of graphical models (see Sec. 2.3.2). The IGS software itself is fully synchronuous, i.e., all tasks are executed in a fixed order in a central application loop. At a certain point of this application loop, the network requests are processed. When this happens, the tracking information is sent to the computer running the *ARGUS* software, and the main AR rendering loop can continue. Therefore, if the main application loop of the image guided surgery device is executed with a low frequency, this also reduces the frame rate of the AR application.

Table 2.1 lists the results of benchmarks measured in the *ARGUS* software. In these benchmarks, the overall frame rate of the system and the time required for the network download of tracking data were recorded. The measured network download delay is the time which passes between sending the request to the IGS system and the reception of the complete data package. In the benchmark session, different configurations of the IGS software were used. The first column of Table 2.1 lists the IGS configurations, which consist of different types of data display. In the "freezed IGS display" setup, a special function of the IGS software was used for disabling the continual update of its graphical display. A single axial volume slice was displayed on the IGS device in the "axial slice shown" configuration. In the "three slices shown" setup, a relatively complex display mode providing the real-time visualization of three volume slices (axial, coronal, and saggittal) was selected.

The benchmark results clearly show the dependency of the AR frame rate on the complexity of the graphical display on the IGS device. A performance of almost 28 fps is achieved in

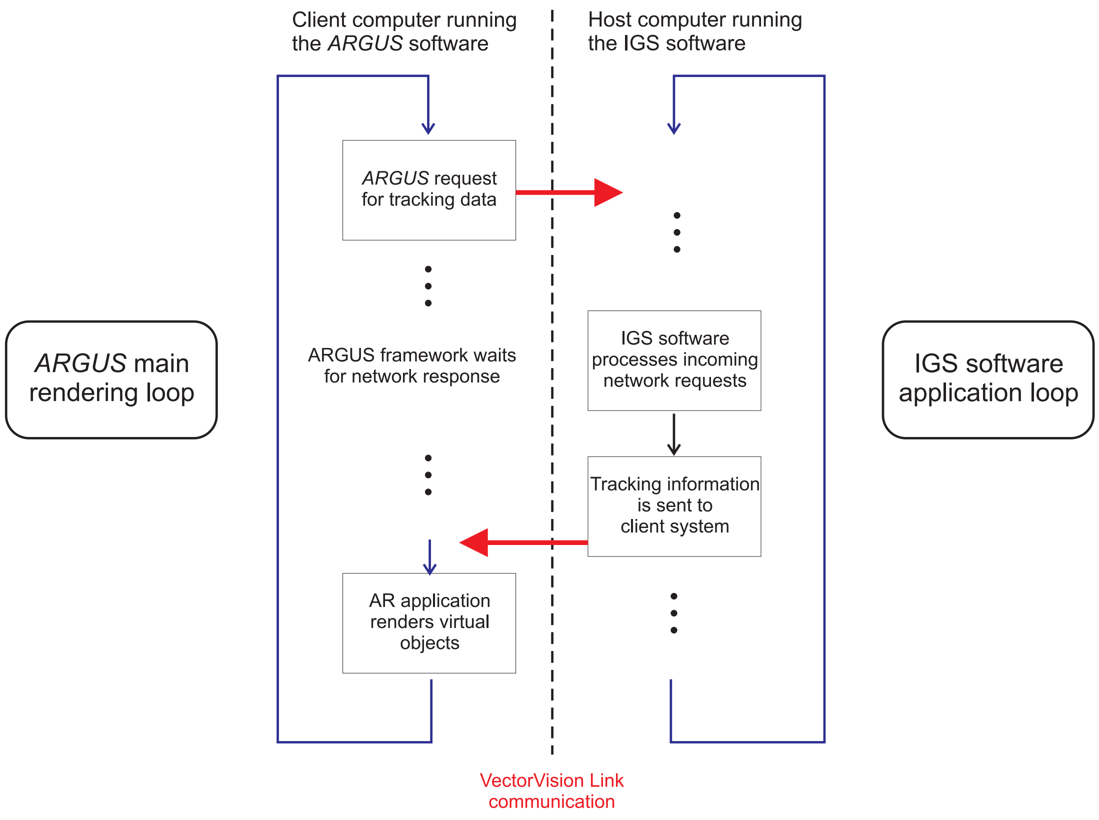

Client computer running the *ARGUS* software
Host computer running the IGS software
*ARGUS* request for tracking data
ARGUS framework waits for network response
AR application renders virtual objects
IGS software processes incoming network requests
Tracking information is sent to client system
*ARGUS* main rendering loop
IGS software application loop
VectorVision Link communication

Figure 2.17: Illustration of the communication flow between the *ARGUS* client system and the image guided surgery device.

the best case (“freezed” configuration), which is close to the theoretical maximum of 30 Hz delivered by the webcam. In the most disadvantageous setup (“three slices”), an insufficient frame rate of only little more than 8 fps was measured. However, in the typical configuration used during the development and operation of *ARGUS*, an average network download time of 58 milliseconds and a resulting overall performance of more than 15 fps were observed. This frame rate is sufficient for an interactive augmented reality application.

The camera tracking provided by the image guided surgery system usually delivers an adequate accuracy. Sporadic visual mismatches between virtual and real scene elements result from the time lag between IGS infrared marker registration, webcam image acquisition, and the generation of the augmented video. Moreover, the accuracy of the overlay depends on the quality of the computed calibration between the webcam and the tracked infrared clamp. Therefore, the one-time calibration step (Sec. 2.3.2) has to be performed diligently. A draw-

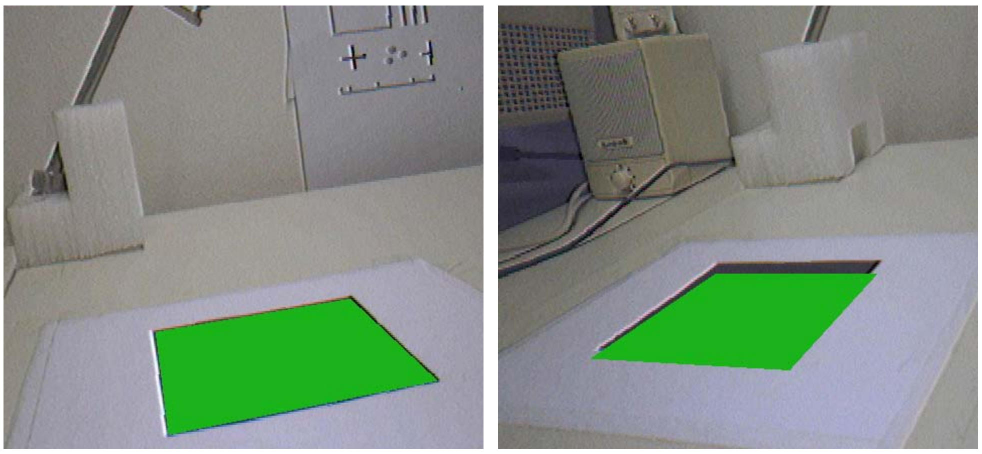
Table 2.1: Average overall frame rates and network download times measured in the *ARGUS* system.

| **Configuration** | **Average AR frame rate** | **Average download time** |
|---|---|---|
| | (fps) | (msecs) |
| Freezed IGS display | 27.99 | 26 |
| Axial slice shown | 15.25 | 58 |
| Three slices shown | 8.38 | 112 |

back of the described tracking method is the fact that the tracking range is limited to the viewing volume of the infrared camera pair of the IGS system. The system is also sensitive to occlusion of the marker clamps from the viewpoint of the infrared cameras. These limitations, however, are the normal restrictions of the image guided surgery device, and the users of this technology are accustomed to them. In practice, a setup of the IGS system is normally chosen which minimizes the occurence of occlusions and places the trackable volume at the optimal location in the operation situs. Other limitations of the infrared camera system which affect the described camera tracking method are the limited angular accuracy of the tracking data and a sensitivity to certain environmental conditions, as described below (see Sec. 2.4).

## 2.4 Model-based Hybrid Tracking for Medical AR

As discussed above, the infrared tracking delivered by the IGS system in combination with the aforementioned calibration step is principally capable of generating a useful overlay of graphical information. However, the overall accuracy of this approach to camera pose estimation is not always flawless. A drawback of the basic *ARGUS* medical augmented reality system is the fact that sometimes discernible tracking errors can occur (see Fig. 2.18). One of the reasons for this is the limited angular accuracy of the delivered tracking data. This is caused by the configuration of the infrared marker clamps used by the VectorVision device, since they consist of only three reflective spheres. This small number of spheres makes it impossible to compensate for localization inaccuracies of individual spheres in the infrared camera view. Moreover, errors can be caused by a number of other factors including an inaccurately performed system calibration, an inadequate patient registration, and a temporarily decreased accuracy of the infrared tracking cameras. The latter can result from environmental conditions like temperature changes or the presence of diffuse daylight in the trackable volume.



(a) Correct graphical overlay (b) Discernible tracking error

Figure 2.18: Comparison of correct and erratic overlay of graphical information. In this example, a virtual green square is rendered over the measured position of an ARToolKit marker (however, tracking is performed based on the infrared tracking cameras of the IGS system). In 2.18b, the virtual marker is visibly displaced from the correct location due to an inaccurate system calibration.

In this section, a new hybrid camera tracking scheme is presented, which aims at reducing the overlay error of graphical objects in medical AR while not requiring any additional equipment [20]. This new tracking algorithm was developed in the context of this thesis and implemented by Michael Eichler in his *Diplomarbeit* (Master's thesis) [80].

### 2.4.1 Related Work on Hybrid Tracking in AR

The combination of different tracking techniques, an approach which is known as *hybrid tracking* or *sensor fusion*, has previously been used in various augmented reality applications. As a very early example, State et al. [185] presented a medical AR system which integrated a magnetic tracker with optical landmark information from the camera image. The objective of this approach can be considered to be somewhat similar to the new system described here, however, they used a dedicated magnetic tracking device which can be problematic when applied in a real medical scenario. Hybrid tracking methods for wide area outdoor augmented reality were described for instance by Piekarski et al. [162] and Caarls et al. [68]. These wide area AR systems use specialized tracking devices like the global positioning system (GPS), which are not suitable for medial applications. You et al. developed a hybrid tracking system for AR, combining a specialized inertial tracker with vision-based pose estimation [211]. Recently, Schwald and Seibert have described the integration of a dedicated infrared camera system and an electromagnetic tracker into the Medarpa medical augmented reality setup [178].

### 2.4.2 Overview of the Hybrid Tracking Method

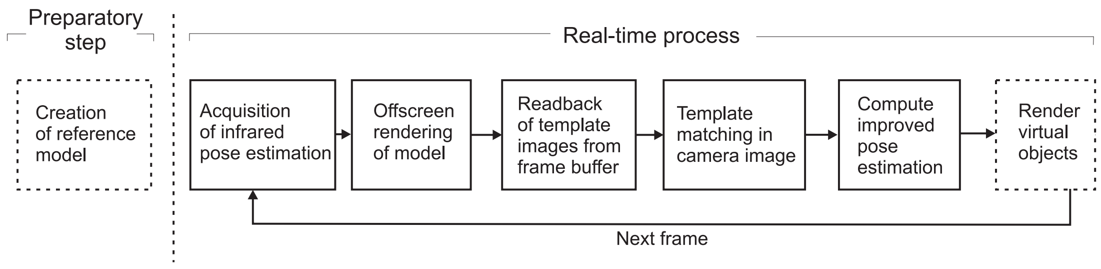

Preparatory step
Real-time process
Creation of reference model
Acquisition of infrared pose estimation
Offscreen rendering of model
Readback of template images from frame buffer
Template matching in camera image
Compute improved pose estimation
Render virtual objects
Next frame

Figure 2.19: Overview of the proposed hybrid tracking algorithm.

A hybrid tracking approach for medical augmented reality was developed. This hybrid tracking system combines an initial pose estimation from the infrared cameras with information from the digital camera image. This way, the advantages of the two basic tracking methods complement each other. The infrared tracking provided by the medical device is stable in the sense that it delivers a pose estimation in practically every frame. Because the infrared cameras are mounted on a movable swivel arm and have a large trackable volume, failures of the infrared tracking due to occlusion or visibility problems rarely occur. The stability of this infrared pose estimation is combined with the improved accuracy of the image-based component. While the image-based method is not able to deliver an initial global tracking, it can compensate errors in the position and orientation estimation of the infrared cameras.

An overview of the method is shown in Figure 2.19. The presented approach is a model-based algorithm. This means that it relies on a geometrical model of a real object in the observed scene. In a preparatory step, this reference model is manually defined by the user.

The model consists of a set of salient feature points defined in 3D together with associated pictures showing the respective portion of the real object.

In the central application loop of the hybrid tracking system, at first the initial pose estimation is acquired from the image guided surgery device. The system then renders a graphical representation of the geometrical model into the currently not visible back frame buffer. For this rendering process, the infrared pose estimation is used as global coordinate system transformation. For each of the previously defined salient feature points, a template image is then read from the back frame buffer. The system looks for a corresponding location in the camera image for each template image, in a search area centered at the projected feature point position. This yields the position of a so-called *correspondence point* in the camera image, as well as a measure for the similarity of this camera image location with the image template.

The pairs of feature point positions and detected correspondence points are then fed into a numerical optimization algorithm. An iterative optimization approach is used, which starts with the infrared pose estimation and incrementally updates the pose parameters in order to minimize the reprojection error of the feature points. This approach can be considered a tightly integrated hybrid method because both the infrared pose estimation and the image-based feature correspondences are used in the same numerical computation. In order to improve the numerical stability of the algorithm, a number of preprocessing steps are performed on the point correspondence data, which are described in Section 2.4.3.

In the remainder of this section, the following nomenclature will be used:

- The salient feature points defined in 3D are called the *reference points* **R**, consisting of individual reference points $r_i = (x_{r_i}, y_{r_i}, z_{r_i})$.
- The set of two-dimensional correspondence points is called **C**, containing the individual correspondence points $c_i = (x_{c_i}, y_{c_i})$.
- The initial pose estimation delivered by the infrared camera system is called *infrared pose*. This pose is represented as a transformation matrix $M_{IR}$.
- The (iteratively refined) pose estimation delivered by the hybrid tracking algorithm is called the *hybrid pose*, stored in a transformation matrix $M_{Hyb}$.
- At the beginning of the algorithm, the reference points are projected into 2D camera image space with the initial infrared pose. These projected reference points are denoted as $p_i = (x_{p_i}, y_{p_i})$.
- In order to make a simplified notation possible, the function $proj(a)$ is used for denoting the projection of a 3D point $a$ into two-dimensional image space. In the *ARGUS* system, the internal camera parameters are determined with the help of the ARToolKit framework. The projection operation then consists of multiplying the 3D point with the camera parameter matrix and subsequently performing the perspective division. Using this notation, the connection between the reference points and the associated projections can easily be formulated as $p_i = proj(M_{IR} \cdot r_i)$.

### 2.4.3 Description of the Algorithm

#### Creation of Reference Model

The newly developed hybrid tracking algorithm requires a geometrical model of some real object in the observed scene. In the current implementation of the system, this reference model is created manually by the user with a separate software tool. The model is defined using a simple process based on a sequence of camera viewpoints. For each viewpoint, the digital camera is placed so that it can take a picture of some portion of the real reference object. The user then indicates salient reference points in the visible portion of the model. The definition of the 3D point positions is done with a special pointer tool which is tracked by the IGS device. A specific rotating pointer tool gesture is recognized by the system and used for determining the position of the salient point. This process is shown in Figure 2.20. (A more detailed description of user interaction in the ARGUS system is given in Section 2.5.)

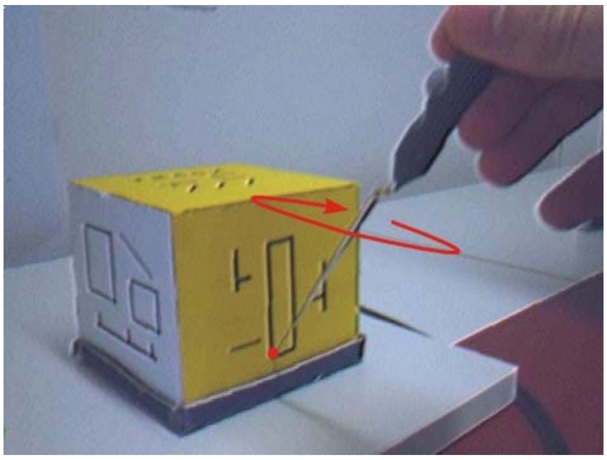


Figure 2.20: Definition of a reference point with a rotating pointer tool gesture. The reference object used here is a cube with artifical high-contrast patterns.

For each defined reference point, a quadratic image template (60 x 60 pixels) centered at the point position is copied from the current camera image. In addition to the 3D point position and the associated image data, the spatial positions of the corners of the image template are also stored. This is necessary so that it is later possible to render the reference model. These corner positions are determined by calculating their locations in the camera image and back-projecting these locations into the 3D world coordinate system. In summary, the following data are stored for each reference point:

1. Reference point location $r_i$
2. Associated template image
3. 3D position of the four corner points

Figure 2.21 shows a graphical representation of the model data acquired for the reference cube which was used during the development of the algorithm.

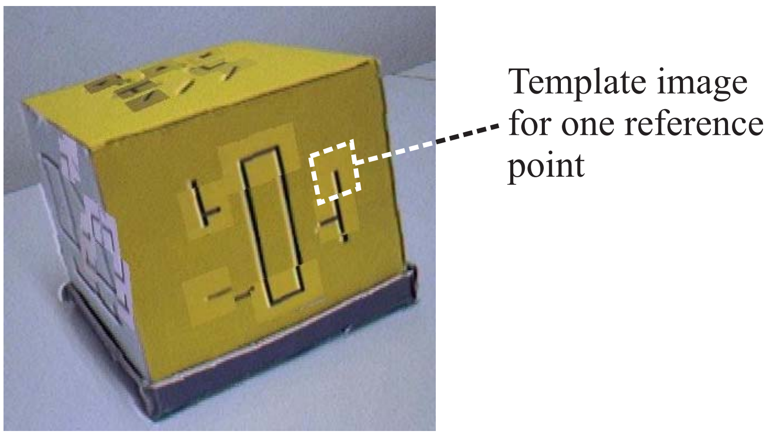


Figure 2.21: Visualization of the model data acquired for the reference cube. In this image, the projected image templates are overlaid over the real camera image.

### Offscreen Rendering of Model

At the beginning of the main application loop of the hybrid tracking system, the current infrared pose estimation for the digital camera, $M_{IR}$, is acquired from the IGS device. (Also see Fig. 2.19, where this step is depicted as the first stage of the realtime process.). Subsequently, the reference model is rendered according to this initial pose estimation. This is achieved by setting the OpenGL transformation matrix such that it reflects the camera pose. For each reference point, a square is then rendered in 3D using the stored image corner positions from the model definition stage. Each reference point square is textured with the corresponding template image acquired during the model definition. Figure 2.22 shows the offscreen representation of the reference cube model used during development.

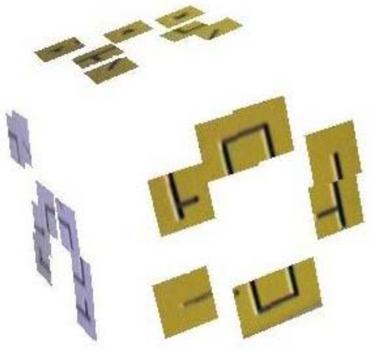


Figure 2.22: The offscreen representation of the reference model of the example cube. Shown here is the final result after all template images were rendered according to the inital infrared pose.

After the textured square for each reference point has been rendered, the actual template image for this point is read back from the frame buffer. A quadratic part of the frame buffer image centered at the location of the projected reference point $p_i$ is used as the template image. During the rendering process, depth buffer tests are disabled, resulting in complete visibility for the last rendered textured square. The principle of reading back quadratic frame buffer areas centered at the projected reference points is illustrated in Figure 2.23.

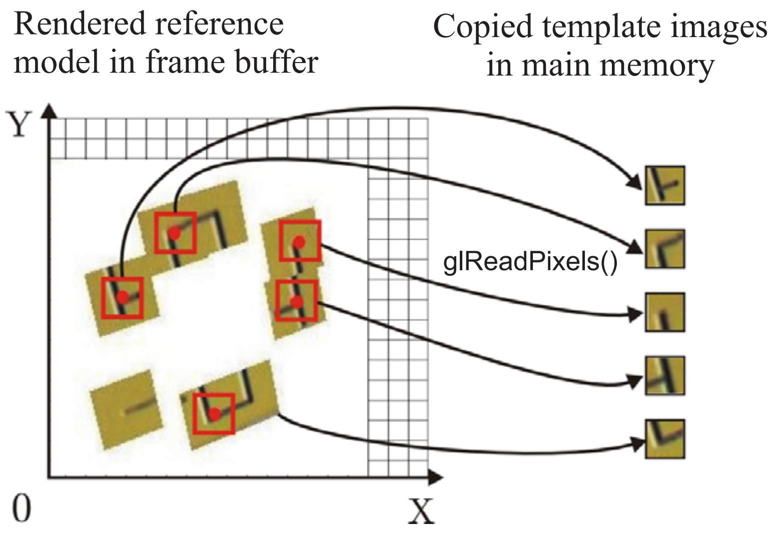

Rendered reference model in frame buffer
Copied template images in main memory
Y
glReadPixels()
0
X

Figure 2.23: After the textured square for each reference point has been rendered, the corresponding projected template image is read back from the frame buffer.

Each final template image is read back into main memory using the OpenGL function `glReadPixels()`. The entire reference model is rendered into the currently invisible back buffer of the OpenGL doublebuffer. This rendered representation is later overwritten by the composed AR frame and is never visible to the user. This step of the algorithm can therefore be considered an offscreen rendering process.

### Template Matching

After the offscreen rendering step, the correspondence point $c_i$ is searched for each projected template image. This is done by defining a search area in the camera image which is also centered at the location of the projected reference point ($p_i$). Figure 2.24 shows a schematic overview of the template matching process. In the figure, $t$ denotes the side length of the quadratic template image, and $s$ is the side length of the search area. Both $s$ and $t$ are user-definable parameters.

Template matching is a common task in image processing, and various different approaches to template matching exist [163]. In the hybrid tracking system described here, the so-called normalized correlation coefficient is used. A publicly available and speed-optimized implementation of this algorithm is provided by the OpenCV computer vision library, which is utilized for the basic image processing tasks in the hybrid tracking system [116]. This template matching method uses single-channel images as input data. Therefore, both the current camera image and the currently considered template image are converted into gray images. In Equation 2.5, $r(x, y)$ is the computed normalized correlation coefficient at the pixel coordinates $(x, y)$ in the camera image. The basis for the computation of the coefficient is a cross-

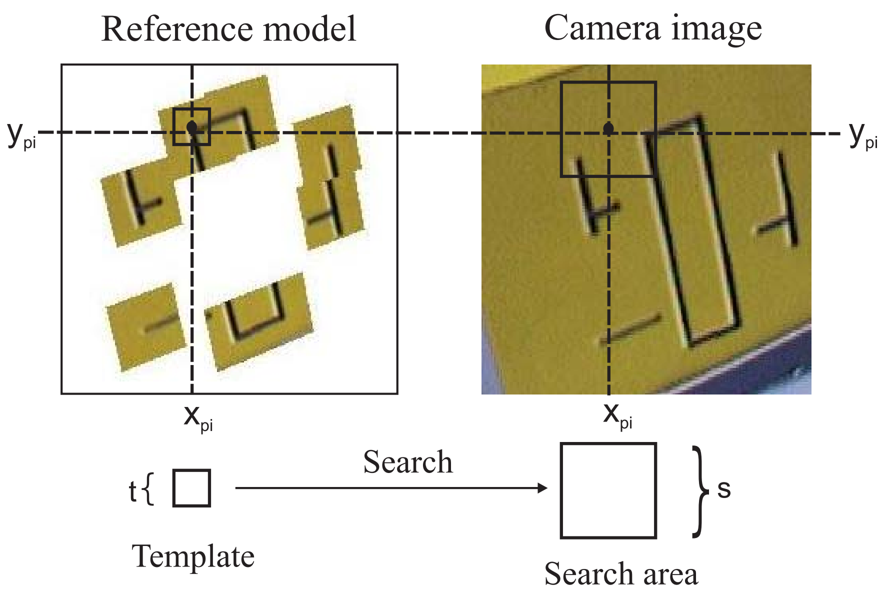


Figure 2.24: Overview of the template matching process. A search area is defined which is centered at the projected reference point, $p_i$. The algorithm then looks for the respective correspondence point $c_i$ in this portion of the camera image.

correlation between $T(x, y)$, which is the pixel intensity of the template image, and $C(x, y)$, which is the pixel intensity of the camera image. As shown in the equation, the result of the correlation is divided by a term that accounts for the total intensity in the considered image area. This way, comparable correlation values are obtained for images of varying brightness.

$$r(x,y) = \frac{\sum_{y'=0}^{t-1}\sum_{x'=0}^{t-1} \widetilde{T}(x',y')\widetilde{C}(x+x',y+y')}{\sqrt{\sum_{y'=0}^{t-1}\sum_{x'=0}^{t-1} \widetilde{T}(x',y')^2\widetilde{C}(x+x',y+y')^2}} \tag{2.5}$$

For each pixel in the search area, the summations in Equation 2.5 are evaluated over the area of the template image, $t \cdot t$. The factors added up in the summations, $\widetilde{T}$ and $\widetilde{C}$, are indirectly derived from the pixel intensities. In order to make the resulting coefficient even more independent from varying brightness in the camera and template images, only the differences of the pixel intensities from the average intensity are considered. An average intensity $\overline{C}$ is computed for the currently regarded search area. Correspondingly, the average intensity $\overline{T}$ is calculated for the template image. The factors used in the computation of the normalized correlation coefficient are then determined as shown in Equation 2.6.

$$\begin{aligned}\widetilde{C}(x,y) &= C(x,y) - \overline{C}\\ \widetilde{T}(x,y) &= T(x,y) - \overline{T}\end{aligned} \tag{2.6}$$

For each reference point, the normalized correlation coefficient is calculated over the entire associated search area. The coefficient specifies the similarity between the region centered at this position in the camera image and the projected template image. The greater the coefficient value, the greater is the similarity between both images. After the computation of the correlation coefficients, the location of the maximum similarity is determined. The result of this search consists of two pieces of information. The main result is the correspondonce point $c_i$, which is considered to be the place in the camera image which corresponds to the projected reference point $p_i$. The second information gained from the search process is the maximum coefficient itself. This maximum coefficient, which is here denoted as the *confidence value* $k_i$, is also stored for later use.

### Numerical Stability

After the correspondence points have been determined for all reference points, the actual computation of the improved camera pose can be performed. This pose computation, however, is embedded into several methods for increasing the stability of the numerical algorithm. At first, inadequate point correspondences are culled from the set of reference points. These are point correspondences with confidence values below a certain threshold, i.e., $k_i < k_{min}$. Here, $k_{min}$ is a constant value, which can be defined by the user. Point correspondences with small confidence values represent invalid template matching results. These can be caused by various circumstances, e.g., if the search area is too small or if the real feature point is occluded in the camera image. Removing these invalid correspondences significantly improves the quality of the hybrid pose estimation process. Values between 0.8 and 0.9 have empirically proven to be good choices for $k_{min}$.

As a second measure for improving the numerical stability of the pose computation, all relevant point coordinates are normalized. This means that they are transformed into a coordinate system in which the average distance of points to the coordinate origin is $\sqrt{2}$ in 2D or $\sqrt{3}$ in 3D, respectively (see [108]). Such a transformation is applied to both the reference points $r_i$ and the correspondence points $c_i$. The effect of this normalization is that during the actual numerical computation, the range of occuring numerical values is relatively small. Therefore, negative effects caused by the limited accuracy of floating point variables are restricted.

Finally, the actual pose computation process is embedded in a random sample consensus algorithm (RANSAC) [88]. The RANSAC approach helps to minimize the impact of invalid point correspondences which have not been removed by the confidence threshold. There are several possible reasons for the occurrence of such invalid correspondences with high similarity values. Among the possible causes is the existence of repeated patterns in the image, excessively large camera movements, or inappropriately defined reference points.

### Pose Estimation

The core of the hybrid tracking system is a pose estimation algorithm based on Newton's method for solving systems of non-linear equations. This pose estimation method is described in detail by Trucco and Verri [195]. The big advantage of this method is that it starts with an initial estimate for the pose. In the system described here, the acquired infrared pose, $M_{IR}$, is used as this initial estimate.

The algorithm uses a representation of pose information which consists of the translation from the coordinate origin, $T = (t_x, t_y, t_z)$, and the orientation expressed in Euler angles, $\Phi = (\phi_x, \phi_y, \phi_z)$. The conversion between a transformation matrix and the representation as $(\Phi, T)$ can be performed with standard linear algebra methods.

Each iteration begins with the computation of the so-called residuals $(\delta x_i, \delta y_i)$. These are the difference vectors between the projections of the reference points according to the current pose estimation and the correspondence points $c_i$. The calculation of the residuals is shown in Equation 2.7. In this equation, each reference point $r_i$ is first transformed according to the current pose parameters, resulting in a transformed reference point $t_i = (x_{t_i}, y_{t_i}, z_{t_i})$. It is then projected into image space, producing a 2D point $q_i = (x_{q_i}, y_{q_i})$.

$$\begin{aligned} t_i &= \Phi \cdot r_i + T \\ q_i &= proj(t_i) \\ \begin{pmatrix} \delta x_i \\ \delta y_i \end{pmatrix} &= q_i - c_i \end{aligned} \tag{2.7}$$

Using Equation 2.7 and the definition of the projection function (see Sec. 2.4.2), the coordinates of the projected points $q_i$ can be differentiated with respect to the translation vector $T$ and the rotation angles $\Phi$. According to [195], this differentiation results in the partial derivatives shown in Equation 2.8 for the translation components $T = (t_x, t_y, t_z)$.

$$\begin{aligned} \frac{\partial x_{q_i}}{\partial t_x} &= \frac{f}{z_{t_i}}, \quad \frac{\partial x_{q_i}}{\partial t_y} = 0, \quad \frac{\partial x_{q_i}}{\partial t_z} = -f\frac{x_{t_i}}{z_{t_i}^2} \\ \frac{\partial y_{q_i}}{\partial t_x} &= 0, \quad \frac{\partial y_{q_i}}{\partial t_y} = \frac{f}{z_{t_i}}, \quad \frac{\partial y_{q_i}}{\partial t_z} = -f\frac{y_{t_i}}{z_{t_i}^2} \end{aligned} \tag{2.8}$$

The partial derivates with respect to the orientation angles $(\phi_x, \phi_y, \phi_z)$ are shown in Equation 2.9. In both equations, $f$ denotes the focal length of the digital camera used in the AR system.

$$\begin{aligned} \frac{\partial x_{q_i}}{\partial \phi_x} &= -\frac{f x_{t_i} y_{t_i}}{z_{t_i}^2}, \quad \frac{\partial x_{q_i}}{\partial \phi_y} = f\frac{x_{t_i}^2 + z_{t_i}^2}{z_{t_i}^2}, \quad \frac{\partial x_{q_i}}{\partial \phi_z} = -f\frac{y_{t_i}}{z_{t_i}} \\ \frac{\partial y_{q_i}}{\partial \phi_x} &= -f\frac{z_{t_i}^2 + y_{t_i}^2}{z_{t_i}^2}, \quad \frac{\partial y_{q_i}}{\partial \phi_y} = f\frac{x_{t_i} y_{t_i}}{z_{t_i}^2}, \quad \frac{\partial y_{q_i}}{\partial \phi_z} = f\frac{x_{t_i}}{z_{t_i}} \end{aligned} \tag{2.9}$$

The value of these partial derivatives is computed for each transformed reference point $t_i$. Using these values, an equation system is then set up for the unkowns $\Delta T = (\Delta t_x, \Delta t_y, \Delta t_z)$ and $\Delta\Phi = (\Delta\phi_x, \Delta\phi_y, \Delta\phi_z)$. $\Delta T$ is the correction for the translation vector of the current pose estimation. Likewise, $\Delta\Phi$ is the correction for the orientation angles of the current pose estimation. These corrections will later be applied to the current pose parameters in order to obtain an improved estimation.

For each correspondence point, the two equations shown in Equation 2.10 are set up. The equations for all point pairs $(t_i, c_i)$ are combined in a single large equation system. This equation system is normally over-determined. A minimum number of five point correspondences was empirically determined in order to compute a useful pose correction. The singular value decomposition is used for solving the equation system [195]. This yields the corrections $\Delta T$ and $\Delta\Phi$ for the current iteration of the algorithm.

$$\begin{pmatrix} \frac{\partial x_{q_i}}{\partial t_x} & \frac{\partial x_{q_i}}{\partial \phi_x} & \frac{\partial x_{q_i}}{\partial t_y} & \frac{\partial x_{q_i}}{\partial \phi_y} & \frac{\partial x_{q_i}}{\partial t_z} & \frac{\partial x_{q_i}}{\partial \phi_z} \\ \frac{\partial y_{q_i}}{\partial t_x} & \frac{\partial y_{q_i}}{\partial \phi_x} & \frac{\partial y_{q_i}}{\partial t_y} & \frac{\partial y_{q_i}}{\partial \phi_y} & \frac{\partial y_{q_i}}{\partial t_z} & \frac{\partial y_{q_i}}{\partial \phi_z} \end{pmatrix} \cdot \begin{pmatrix} \Delta t_x \\ \Delta \phi_x \\ \Delta t_y \\ \Delta \phi_y \\ \Delta t_z \\ \Delta \phi_z \end{pmatrix} = \begin{pmatrix} \delta x_i \\ \delta y_i \end{pmatrix} \tag{2.10}$$

At the end of each iteration, the computed corrections are applied to the current pose estimation. The determined translation correction $\Delta T$ is subtracted from the translation vector $T$ of the current pose, i.e., $T \leftarrow T - \Delta T$. Likewise, the orientation correction $\Delta\Phi$ is applied to the current orientation. This is done by multiplying the rotation matrix corresponding to the current orientation with the corrections for the individual axes. This computation is shown in Equation 2.11. In this equation, $M_\Phi$ is the rotation matrix representing the orientation $\Phi$, and $R_{\{x,y,z\}}(\alpha)$ denote matrices corresponding to a rotation around one of the coordinate axes.

$$M_\Phi \leftarrow M_\Phi \cdot R_x(-\Delta\phi_x) \cdot R_y(-\Delta\phi_y) \cdot R_z(-\Delta\phi_z) \tag{2.11}$$

The pose estimation algorithm can thus briefly be summarized as follows: Starting with $M_{IR}$ as initial estimate, in each iteration **1.** compute the residuals $(\delta x_i, \delta y_i)$, **2.** for each point correspondence, calculate the partial derivatives and construct the coefficient matrix, **3.** set up the complete equation system and solve it for $\Delta T$ and $\Delta\Phi$, and **4.** update the pose estimation with the obtained corrections. This is repeated until either a maximum number of iterations is reached or the average length of the residuals becomes smaller than a threshold. In the implementation, a maximum number of 30 iterations is used, which has empirically proven to produce good results. The finally obtained pose estimation is used as improved hybrid camera pose $M_{Hyb}$ in the AR system.

### 2.4.4 Results

The presented hybrid tracking system was developed and tested with the aforementioned artificially textured cube object as reference model. Good results were obtained with the hybrid scheme, and a significantly improved pose estimation was achieved under most circumstances. Figure 2.25 demonstrates the effect of the hybrid tracking system. It clearly shows that the cube geometry rendered with the hybrid pose estimation corresponds significantly better to the location of the real cube in the camera image. The effect of the hybrid scheme on the projections of the reference points is shown in Figure 2.26. Again, the reference points projected with the hybrid pose are significantly closer to their real counterparts (locations of high-contrast corners).

Several experimental test runs were performed. Table 2.2 lists the parameters used for one typical test run. In this case, a reference model was used which consists of 23 reference points defined from three camera viewpoints. The experiment spanned a duration of 182 frames, during which the camera was moved relative to the example cube, but remained roughly pointed at the object.

Table 2.3 compares the tracking accuracies achieved with the infrared pose and the hybrid tracking. The pixel displacement listed in the table is the distance between the projected reference points and the associated correspondence points. For the infrared pose, this is $||p_i - c_i||$,

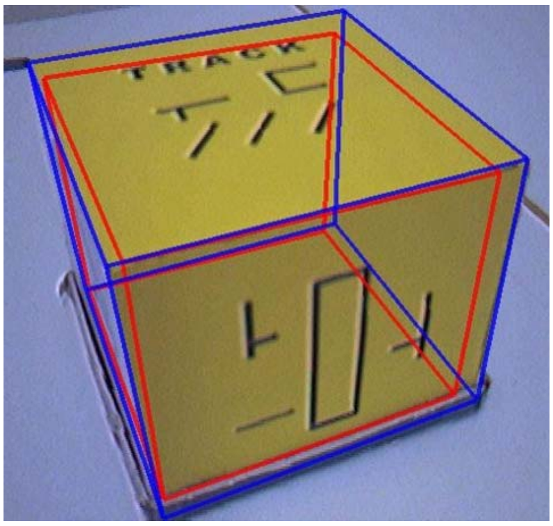


Figure 2.25: Visualization of the effect of the hybrid tracking approach. The red (brighter) wireframe was rendered with the initial infrared pose, the blue (darker) wireframe with the improved hybrid pose.

| | |
|---|---|
| Template side length $t$ | 16 pixels |
| Search area side length $s$ | 60 pixels |
| Confidence threshold $k_{min}$ | 0.9 |
| Number of reference points (defined from 3 viewpoints) | 23 |
| Test duration | 182 frames |

Table 2.2: Parameters of the experimental test run.

for the hybrid pose $||proj(M_{Hyb} \cdot r_i) - c_i||$. As shown in the table, the minimum average displacement per frame is significantly smaller for the hybrid pose (2.98 pixels) than for the infrared pose (6.4 pixels). Moreover, the measured overall average displacement is more than 30% less with hybrid tracking (7.89 pixels) than with pure infrared tracking (11.74 pixels).

It has to be noted that it is the default behaviour of the hybrid tracking system to revert to the infrared pose estimate if the hybrid pose is considered to be invalid. This is the case if the average reference point displacement is too large. An invalid pose estimation can be caused by adverse environmental circumstances. These include excessively fast camera movements or rapid changes in the environment lighting, which cause the digital camera to deliver useless images. Another possible reason is a situation in which the reference object is not visible or mostly occluded in the camera image. In the experiment, invalid poses were computed for three frames. These frames were also included in the statistics shown in Table 2.3.

As shown in Table 2.3, the hybrid tracking system reduced the frame rate from more than 20 fps to 13.5 fps. Table 2.4 contains an analysis of the average runtime of the individual algorithm steps during the experiment. The major part of the hybrid pose estimation algorithm is required for the template matching step.

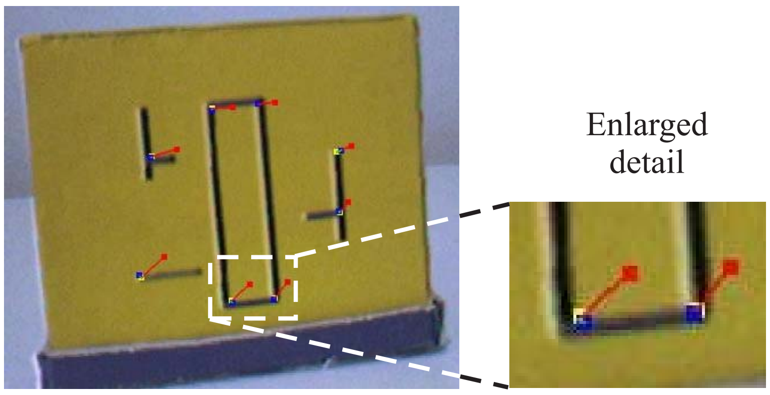


Figure 2.26: The effect of the hybrid tracking scheme illustrated for the reference points. The red dots were projected with the initial infrared pose and correspond to the $p_i$. They are connected with red lines to the blue projections of the reference points when the improved hybrid pose ($M_{Hyb}$) is used. (Also partially visible as yellow dots are the correspondence points $c_i$.)

| | **Infrared pose** | **Hybrid pose** |
|---|---|---|
| Min. Ø displacement (average per frame) | 6.4 pixels | 2.98 pixels |
| Max. Ø displacement (average per frame) | 18.94 pixels | 12.88 pixels |
| Overall Ø displacement (average of all frames) | 11.74 pixels | 7.89 pixels |
| Overall Ø frame rate | 20.7 fps | 13.5 fps |
| Number invalid frames | - | 3 |

Table 2.3: Results of the experimental test run.

| | | |
|---|---|---|
| Offscreen rendering | 17.86 msecs | (27%) |
| Template matching | 36.92 msecs | (57%) |
| Pose estimation | 10.18 msecs | (16%) |

Table 2.4: Runtime analysis of the individual algorithm steps.

### 2.4.5 Discussion

The described hybrid tracking algorithm is capable of significantly improving the accuracy of graphical overlays in video see-through AR. It works stably expect in the case of adverse environmental conditions (see Sec. 2.4.4). However, the system can revert to the infrared pose if an invalid hybrid pose was computed. The hybrid tracker, which uses information from the camera image, has the typical limitations of vision-based systems. A low quality of the digital image, an inadequate definition of the reference model, excessive camera motions, or too much occlusion in the image can have a negative impact on the tracking performance. However, empirically it was found that the hybrid pose estimation system delivered useful output most of the time in the performed experiments.

The current implementation of the algorithm reduces the overall frame rate of the system. The most computationally complex algorithm step is the readback of template images using `glReadPixels()` and the subsequent template matching. This part of the method could be sped up by utilizing the programmability of modern GPUs. With appropriate fragment programs, tasks like template matching could be offloaded to the GPU, eliminating computations on the CPU and costly buffer readbacks.

The main drawback of the current implementation is the required manual definition of the reference model. A (semi)automatic acquisition of reference objects is an important topic in the future work. Moreover, the system should be tested in a more application-specific environment. Possibly, an adjusted tracking strategy could prove useful for medical scenarios, e.g., by using intraoperative registration fiducials as optical landmarks.

The current realization of the presented system is still in an experimental state, but it demonstrates the feasibility and usefulness of the approach. Medical application scenarios can benefit significantly from an improved accuracy of the camera pose estimation.

## 2.5 User Interaction in the ARGUS System

One important challenge for any kind of augmented reality system is to provide useful facilities for user interaction. These should usually include methods for triggering application-specific actions, for the selection or manipulation of virtual objects, and for the definition of points or more complex shapes in space. In medical augmented reality, user input is often required for changing the rendering parameters for medical data visualization and for displaying and manipulating operation plan information. Many different techniques for user interaction in AR have been proposed. Most of these are based on specialized hardware, e.g., magnetic trackers or 3D input devices. Specialized interaction devices for virtual reality were for instance described by Fröhlich et al. [92] and Bernstein et al. [53]. Wormell and Foxlin have given an overview over some recently developed dedicated devices for user input in VR/AR [209]. However, as stated above, the use of such dedicated system components can be problematic in medical applications (see Sec. 2.2.1).

In the context of this thesis, a novel method for user interaction in medical AR was developed [9, 13]. This new approach is based exclusively on the information delivered by the image guided surgery system. The pose data of tracked surgical tools is processed in order to recognize a set of basic "gestures", i.e., simple user interaction elements. The *ARGUS* framework uses these gestures to implement a flexible, configurable menu system. This system makes it possible to trigger actions by performing a "click" at a certain position. These clicks can be executed in immediate proximity to the patient. Moreover, the user can define positions and freely drawn shapes in three-dimensional space. All of these interactions, which are performed with untethered passive input tools tracked by the IGS device, are supported without requiring any kind of additional hardware. Figure 2.27 shows an example of a simple operation plan element created with the new user interaction method.

The menu system described in this thesis could even be used as a replacement for existing user interfaces provided by medical devices. As in the case of the VectorVision system (see Sec. 2.3.1), these are usually based on touchscreens or conventional screens with mouse input. Unlike such traditional user input systems, the new immersive approach in *ARGUS* works in immediate proximity to the patient and does not require the user to shift the attention focus to a screen built into a medical device.

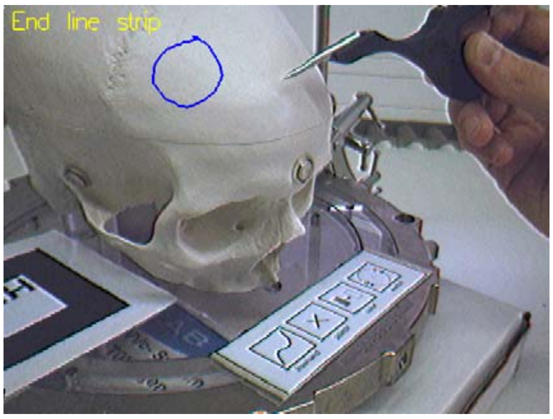


Figure 2.27: Example drawing created directly on the plastic skull. The tracked tool used for interaction can be seen in the top right part of the image. On the bottom, the row of four menu “icons” is visible.

The IGS-based user interaction and menu system developed in the context of this thesis has been filed for patent [40].

### 2.5.1 The New User Interaction Method

A new technique for providing comprehensive user interaction based on the information delivered by the image guided surgery system was developed. The IGS device is capable of tracking multiple infrared marker clamps simultaneously. In order to utilize the infrared tracking for user interaction, one of the instrument clamps is attached to a pointer-like tool, as illustrated in Figure 2.28. Alternatively, the standard pointer tool supplied with the IGS system can be utilized. The tool is used for triggering actions by indicating menu markers located at previously defined positions and for the definition of points or free formed shapes in 3D.

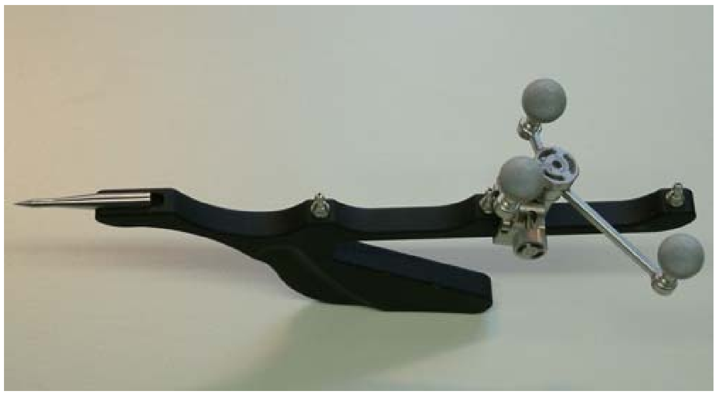


Figure 2.28: Example of a tracked tool used for interaction.

The *ARGUS* system continually acquires the position and orientation of tracked tools from the image guided surgery device. It then looks up the user-defined interaction tool, which is identified by a unique name string. Consecutive tool positions and orientations are compared in order to detect basic gestures and the triggering of menu actions.

### Basic Gestures

The new user interaction system supports two basic gestures. One is the so-called "still click", which is detected when the movement of the tool is below a given threshold for a certain amount of time. The still click is easy to execute, but it is also often triggered inadvertently, e.g., when the tool is put down.

The "angle click" also requires the position of the tool to remain practically constant for a certain duration. However, additionally the direction vector of the tool has to change continually during that period. The position reported by the IGS system is always that of the tip of the tracked instrument. Therefore, an angle click is executed when the user holds the tip of the instrument at a certain point while rotating the instrument. The angle click is significantly more complex to perform and thus is rarely triggered unintentionally. Figure 2.29 illustrates the tool motion necessary for triggering an angle click. Note that the angle click gesture is also used in the model definition phase of the hybrid tracking algorithm discussed in Sec. 2.4.3.

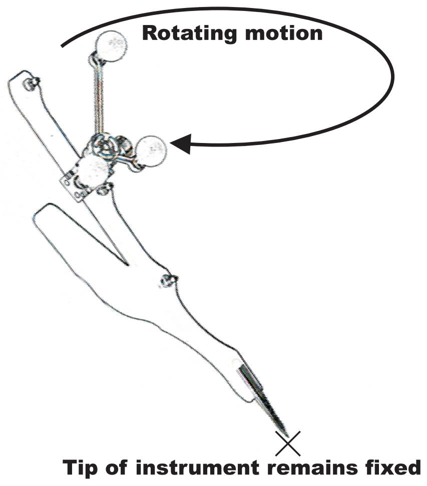


Figure 2.29: Illustration of the "angle click" user interaction gesture.

The following algorithm was developed for detecting tool gestures. In every time step, the position $pos_t$ and the direction vector $dir_t$ of the tracked interaction tool are received from the IGS system. The software permits a maximum tip movement of $threshold_{pos}$ per time step. Since the coordinate system of the infrared camera is calibrated to have a unit length of one millimeter, $threshold_{pos}$ is also defined in that unit. A typical value for $threshold_{pos}$ is less than one millimeter. A second threshold parameter, $threshold_{dir}$, determines the minimum change of the direction vector required for a valid angle click step. The number of consecutive time steps during which the respective gesture conditions have to be fulfilled is given as parameter $clickDuration$. The threshold and click duration values can be configured at run-time.

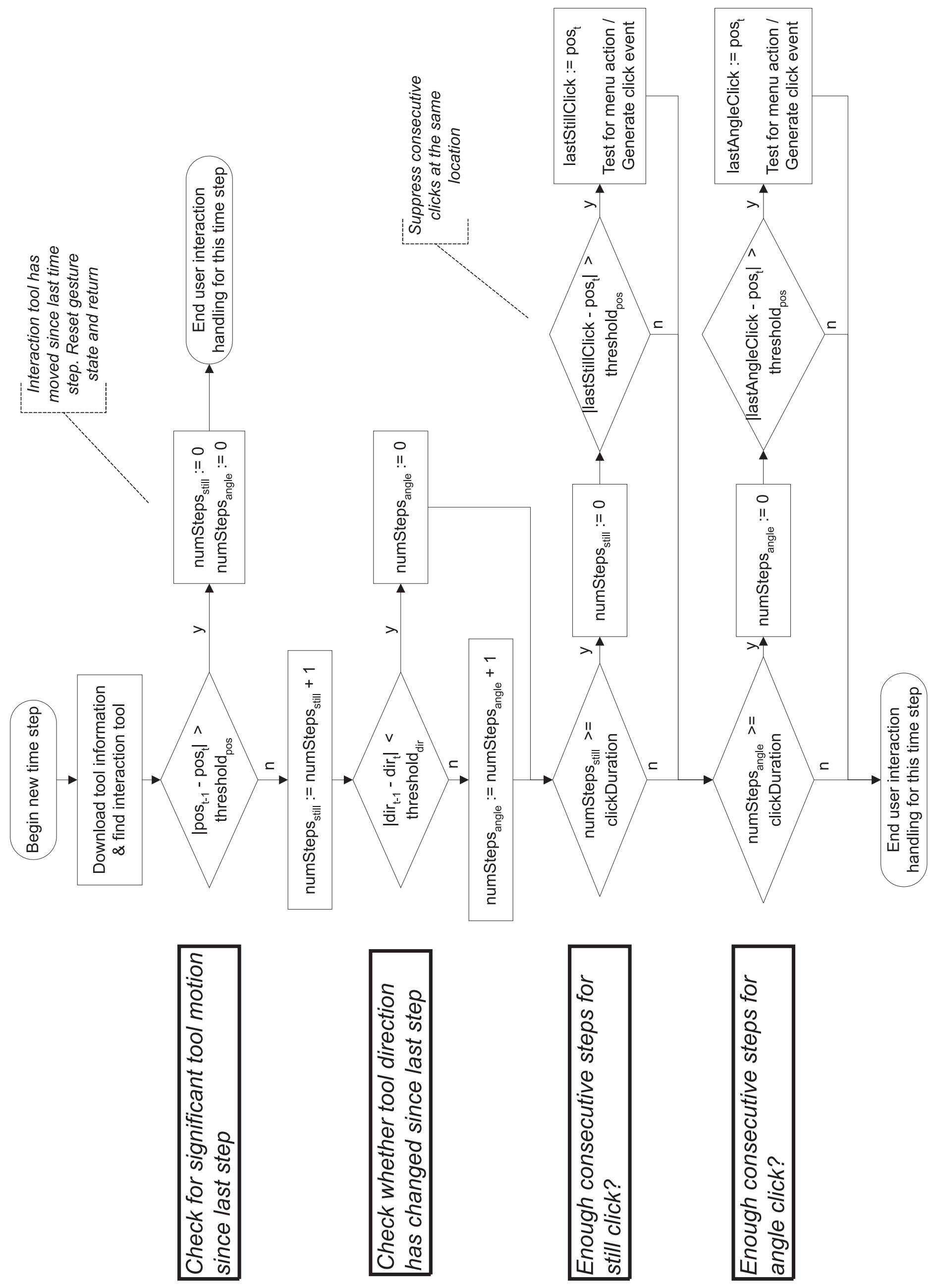

Begin new time step
Download tool information & find interaction tool
|pos_{t-1} - pos_t| > threshold_pos
y
n
numSteps_still := 0
numSteps_angle := 0
Interaction tool has moved since last time step. Reset gesture state and return
End user interaction handling for this time step
numSteps_still := numSteps_still + 1
|dir_{t-1} - dir_t| < threshold_dir
numSteps_angle := 0
numSteps_angle := numSteps_angle + 1
numSteps_still >= clickDuration
numSteps_still := 0
|lastStillClick - pos_t| > threshold_pos
lastStillClick := pos_t
Test for menu action / Generate click event
Suppress consecutive clicks at the same location
numSteps_angle >= clickDuration
numSteps_angle := 0
|lastAngleClick - pos_t| > threshold_pos
lastAngleClick := pos_t
Test for menu action / Generate click event
End user interaction handling for this time step
Check for significant tool motion since last step
Check whether tool direction has changed since last step
Enough consecutive steps for still click?
Enough consecutive steps for angle click?

Figure 2.30: Program flow for updating the gesture state and generating click events for a single time step.

Figure 2.30 shows the program flow chart of the click detection algorithm for a single time step. After the tool information has been downloaded, the movement of the tip is checked. If it is above $threshold_{pos}$, the current gesture state is reset, and the algorithm stops. This means that any ongoing click period is interrupted by the tool motion, and any tool gesture can only begin in a later time step. If the movement is small enough, the counter $numSteps_{still}$ is incremented. $numSteps_{still}$ counts the number of time steps without significant tool motion and is used for triggering still clicks. Subsequently, the current direction vector is compared to the tool direction measured in the last time step. If the direction change is above $threshold_{dir}$, the counter variable $numSteps_{angle}$ is incremented. The value of $numSteps_{angle}$ is used for detecting angle clicks.

After the current changes in tool position and direction have been considered, the conditions for triggering a click event are checked. If $clickDuration$ time steps without significant motion have passed, a still click has been executed. The algorithm then checks whether the current tool position is the same as the position recorded during the last still click. This position is stored in the vector $lastStillClick$. If the locations of the last and the current click are very close, the user interaction event is suppressed. Otherwise, a still click event is generated for further processing by the software, and the current tool position is saved in $lastStillClick$. If the application provides support for menu interaction, the click event is processed by the menu system (see below).

Finally, the algorithm checks whether an angle click has been executed. If the conditions of an angle click were continuously met for $clickDuration$ time steps, the location of the current and the last angle click are compared. In case the current tool position is too close to the last angle click, which is stored in the vector variable $lastAngleClick$, no click event is generated. Otherwise, a user interaction event is sent to the application and, optionally, to the menu system.

It has to be noted that using this method, a still click event is always generated simultaneously or right before an angle click. The application logic is assumed to account for this fact by filtering click events according to the semantics of the current top-level user interaction sequence. The algorithm could easily be modified to operate exclusively in either a still click mode or a separate angle click mode.

### Menu System

In addition to the basic gestures recognition, the second main innovation of the *ARGUS* user interaction system is a method for implementing a configurable menu using the basic gestures. Markers, which have icons for the menu items printed on them, are placed inside the trackable volume of the infrared camera. The user can then activate a menu action by performing a click with the interaction tool on one of the markers. Figure 2.31 shows an example of a simple menu, which is used by the demonstration application for patient drawings (see Section 2.5.2).

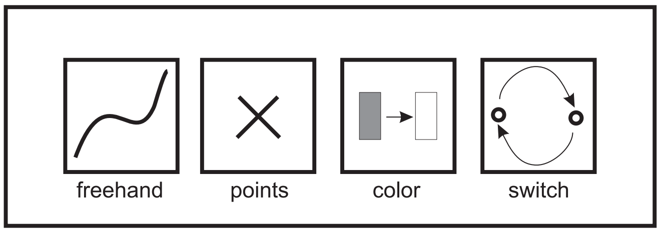


Figure 2.31: AR menu markers used by the patient drawing example application.

The *ARGUS* software contains an editor for the interactive definition and modification of menu items. The data structure for each item consists of the position of the center of the marker, a radius, a name, and a description string. In the editor, the marker position is defined by a click gesture with the interaction tool. The other data are entered with a graphical user interface. Table 2.5 lists the attributes for a menu item and their data types.

Table 2.5: Attributes of a menu item.

| Attribute | Data Type |
|---|---|
| Center position | 3 float values (x,y,z) |
| Radius | float |
| Name | string |
| Description | string |

An application using the menu system can be configured to react to still clicks or angle clicks for menu actions. When a click event of the respective type is generated by the basic gestures algorithm (see Fig. 2.30), it is analyzed by the menu system. The click location is compared with the position of each menu item. If the distance between the click location and the center of a menu item is smaller than its radius, a menu action event is generated. The menu event is parameterized with the name of the menu item and can thus be easily interpreted by the application. Note that due to the comparison of the Euclidean distance with a radius, the real marker is approximated by a sphere. However, this distance criterion could easily be modified to be sensitive to a shape which more closely resembles the physical appearance of the marker. Moreover, in practice, the sphere criterion works reliably and intuitively in most cases.

### 2.5.2 Example Application: Operation Plan Drawings

An example application has been implemented for demonstrating the usefulness of the new *ARGUS* interaction method. Using this software, points and free formed line strips can be drawn directly on the patient. This way, operation plan elements or related visual aids can be created interactively. The application uses the menu items shown in Figure 2.31.

The following mechanism is provided for creating free formed line strips. The software continually waits for menu action events. Once the menu event "freehand" is received, a special drawing mode is initiated. From that moment, the software waits for the next still click. After the still click event has been triggered, the line strip is recorded. Every new position of the tip of the interaction tool is stored and added to the geometry of the line strip. This continues until another still click event is received. The click then ends the interactive drawing of the line strip.

Figure 2.32 illustrates this user interaction sequence. The sequence can be easily and intuitively perfomed by the user. After clicking on the menu item, the interaction tool has to be moved to the intended starting point of the drawing. Then the user has to wait for a short period. Since the basic gestures algorithm suppresses consecutive clicks at the same location (see Fig. 2.30), the line strip never inadvertently begins at the menu item. The drawing is created in a continuous motion. When the drawing is complete, the user again has to wait for a short while in order to leave the drawing mode. This way the entire functionality is controlled without any additional hardware, only using the tracking information of the interaction tool.

The application also contains a mode for setting individual points. When the respective menu action event has been triggered, the software waits for the next still click. A point is then generated at the location of the click. In addition to the line strip and point drawing modes, two utility functions are supported. The menu action "color" changes the current color by

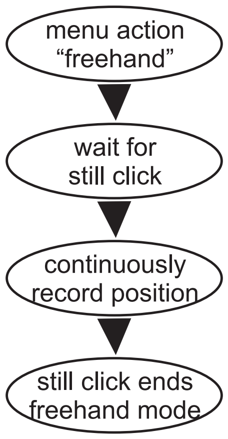


Figure 2.32: Interaction sequence for drawing a free formed line strip in the demonstration application.

traversing a preset color palette. The next point or free formed line strip to be generated is then displayed in the new color. By selecting the menu action “switch”, the user can cycle through several drawing slots. Only one of several drawings is displayed and can be edited at a time. The switch command advances to the next stored drawing or displays nothing, if the slot has not been used yet. Altogether the demonstration software shows that the lightweight interaction method in *ARGUS* can provide all the functionality required for a useful drawing application in an easy-to-use and intuitive way.

In the actual experimental setup, the four menu items are laid out in a row as shown in Figure 2.33. However, it would be possible to place the items freely within the trackable volume of the infrared camera due to the configurability of the menu. A standard XML file contains the description of the menu items and their placement.

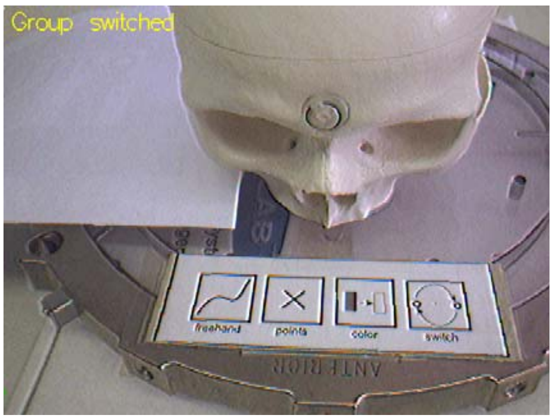
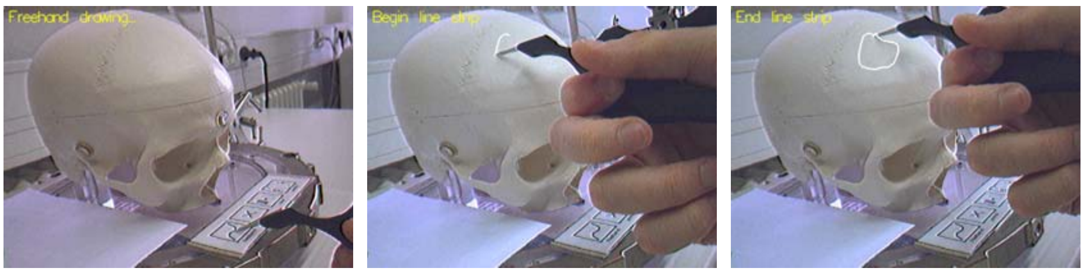


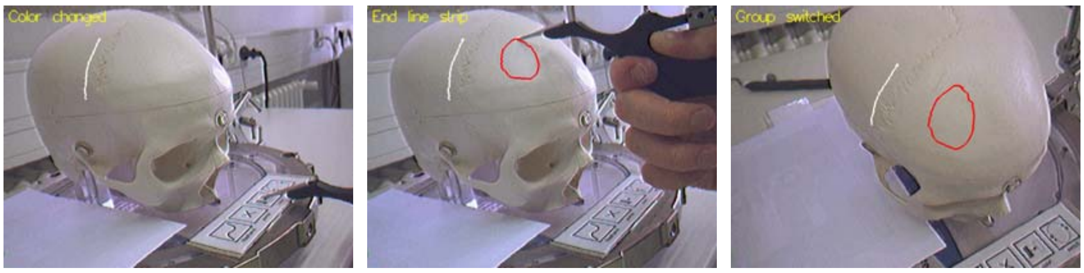
Figure 2.33: Real menu used by the user interaction example application.

Figure 2.34 shows a real-life example for creating a single line strip drawing. The drawing mode is actived in Fig. 2.34a by a still click on the respective menu item. The following click at a different location determines the starting point of the line strip (see Fig. 2.34b). The freely drawn line strip is then extended until the interaction tool again remains still for a period of time (see Fig. 2.34c).



(a) Activation of drawing mode (b) After initial still click (c) End of interaction

Figure 2.34: Real-life example for the creation of a free formed line strip. The interaction mode is activated by a still click on the "freehand" menu item. A second still click begins the drawing, and a third click ends it. In the top left corner of each image, the name of the last detected user interaction is displayed.



(a) Change of color after white line has been drawn (b) Different color used for circular line strip (c) Top view of scene

Figure 2.35: The example application provides a menu item for changing the current drawing color. Figure 2.35c illustrates that the created drawings are three-dimensional.

In Figure 2.35, the effect of the color change action can be seen. Figure 2.35a shows the augmented reality display after a white line strip has been drawn. The user then activates the color change item using a still click. This causes the next line strip to be rendered in a different color (see Fig. 2.35b). Due to the fact that the tracking information delivered by the image guided surgery system contains full spatial pose information, the drawings created with the user interaction system are three-dimensional. This is illustrated in Figure 2.35c, which shows the scene with the virtual drawings from a different point of view.

The described user interaction method in *ARGUS* including the menu system does not negatively affect the overall performance of the AR application. Since the tracked tool data are continually requested from the IGS system for tracking the webcam anyway, the amount of data transferred over the network does not increase.

## 2.6 Occlusion Handling

As described in Sections 2.3 and 2.4, *ARGUS* achieves a spatially correct overlay of graphical information by utilizing an IGS device for camera tracking. The augmented reality application can simply define graphical models representing additional medical information and send the rendering primitives to the standard OpenGL pipeline. The three-dimensional graphical data are then displayed using conventional rasterization methods. These standard graphics algorithms are readily available, easy to use, and fast. The produced graphical output, however, often lacks visual realism. One particular problem caused by the simple overlay of rendering primitives over the background camera image is the lack of correct occlusion handling. This means that the mutual occlusion between real and virtual objects is not represented correctly. It is therefore difficult for the observer to assess the actual depth relationships in the augmented scene.

Figure 2.36 illustrates a typical case of an ambiguous AR image due to the lack of occlusion handling. In both images, the virtual graphical model of a tracked tool is overlaid over the camera image. As described in Section 2.3.4, the graphical tool model is attached to a real tracked instrument held by the user. In Figure 2.36a, the real tool is located between in the plastic skull and the camera. Hence, the resulting image shows the correct depth relationship because the graphical tool occludes the plastic skull. By contrast, in Figure 2.36b, the real instrument is behind the skull phantom. However, the virtual tool model is still rendered over the camera image and appears to occlude the plastic skull. It is therefore not possible for the observer to decide whether the graphical representation of the tracked instrument is supposed to be in front of or behind the real skull phantom.

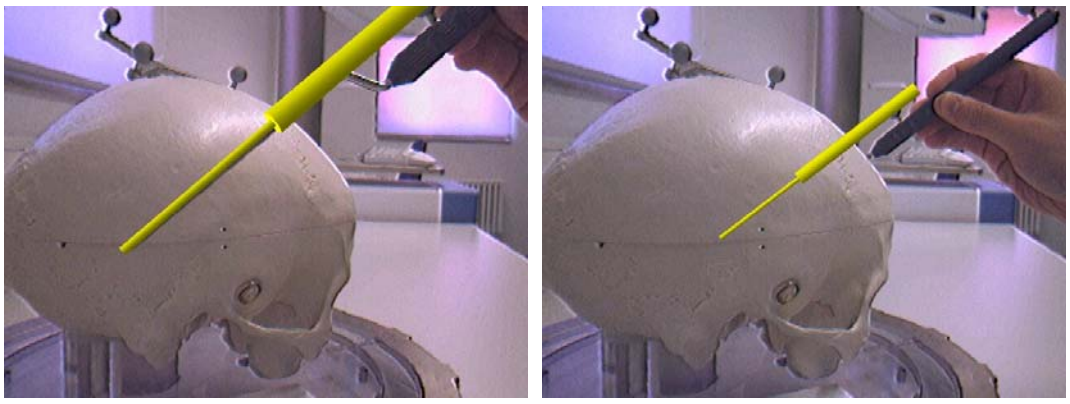

(a) Tracked tool in front of plastic skull (b) Tracked tool behind plastic skull

Figure 2.36: Incorrect representation of depth relationships in the AR scene. Due to the lack of occlusion handling, the graphical tool model occludes the plastic skull both in Fig. 2.36a and in Fig. 2.36b.

The different types of occlusion which can occur in an augmented reality image are listed in Table 2.6. The table distinguishes between real and virtual objects as occluding and occluded object types. A correct image is automatically generated for real objects occluding each other (case 1A) due to optical occlusion. Virtual objects which are supposed to occlude real objects (case 1B) are also intrinsically displayed correctly because they are rendered over the camera image. The standard Z-Buffer test in the rendering pipeline leads to a correct display of depth relationships within the set of virtual objects (case 2B). (The original Z-Buffer algorithm was described in [70, 188].)

The relevant new problem which has to be handled in an augmented reality system is case 2A. Here, virtual objects should be occluded by real objects. This introduces a difficult challenge, which cannot be solved in the general case. In order to handle the occlusion of virtual objects by real objects, the shape and depth of the real objects in the camera image have to be known. Two general approaches to solving this problem exist. In *static occlusion handling*, the 3D geometry of real objects in the actual environment is manually defined before the runtime of the AR system. *Dynamic occlusion handling* tries to resolve occlusion in AR for unknown real objects. The latter approach usually requires relatively complex and computationally expansive image processing algorithms. Both approaches can only partially solve the occlusion handling problem in most cases.

| Occluder / Occluded object | Real objects | Virtual objects |
|---|---|---|
| Real objects | 1A) Optical occlusion | 1B) Graphical primitives overlaid over camera image |
| Virtual objects | 2A) **Occlusion handling (static / dynamic)** | 2B) Graphics hardware (Z-Buffer test) |

Table 2.6: Different types of occlusion occurring in an AR scene. In this table, all possible combinations of occluding object type and occluded object type are listed. For each combination, the table describes how the corresponding occlusion case is resolved. (Adapted from [127].)

In the context of this thesis, a specialized occlusion handling method for medical augmented reality was developed. This new approach solves the occlusion problem for the application case illustrated in Figure 2.36. The new algorithm can handle the occlusion of virtual objects by the anatomy of the patient. Since the geometry of the relevant part of the patient's anatomy is available as a volume dataset, the method is a special case of static occlusion handling.

### 2.6.1 Related Work on Occlusion Handling in Augmented Reality

Several researchers have worked on the detection and handling of occlusion in augmented reality. Breen et al. have suggested a model-based approach to handling occlusion in augmented reality [64]. As mentioned above, the use of geometric models of known real objects for detecting the occlusion of virtual objects is called *static occlusion handling*. These geometric models, which have to be acquired or manually modelled before the runtime of the AR system, are often also called *phantom models*[2]. An example of static occlusion handling is shown in Figure 1.2a. This method of detecting and handling occlusion in augmented reality has been used and extended in many AR systems. An extension of static occlusion handling for determining how virtual objects are hidden by the user's body was described by Fuhrmann et al. [93]. Their system combines a previously acquired geometric model with positional data from a tracking system.

[2]Note that in this thesis, the term *phantom* is also sometimes used to denote the plastic skull mock-up in the experimental AR system setup.

*Dynamic occlusion handling* does not require geometric descriptions of real objects in the environment. In dynamic occlusion handling, image processing and computer vision techniques combined with certain assumptions or partial information about the real environment are used in order to detect occlusion. The dynamic approach can therefore deal with occlusion in AR in the more general case, especially in (mostly) unknown environments. The drawbacks of dynamic occlusion handling, however, are increased computational costs, a more tedious and complex algorithm development, and often a reduced stability of the occlusion detection. Berger has described a method for resolving occlusion when overlaying virtual objects over recorded video sequences [52]. This algorithms tracks the contours of real objects in image space in order to compute an occlusion mask. Later, Lepetit and Berger presented another approach to handling occlusion in off-line augmented reality [133]. In this system, the user manually defines the occlusion boundary of a real object as well as relevant key frames in the stored video sequence. The algorithm then automatically tracks the occlusion boundary, which results in a correct rendering of occluded graphical objects. The author of this thesis has also previously described an algorithm for detecting dynamic occlusion in front of planar, textured background objects [26]. This method tracks salient features in the background texture and performs a comparison between camera image pixels and the projected texture data in order to compute an occlusion map. A similar, but more advanced, algorithm was described by Lin et al. [135]. In their system, a model of the real background is acquired in a semi-automatic procedure from the video stream. This information is then used as the basis for an image-based tracking method as well as for occlusion handling by means of background-foreground segmentation.

Approaches for detecting occlusion in a stereo camera AR system have also been investigated. In some cases, an attempt to solve the occlusion problem using depth information delivered by stereo matching is made [120, 207]. The method developed by Gordon et al. [100] can correctly render interaction devices into the scene.

### 2.6.2 Static Occlusion Handling based on Volumetric Datasets

In the context of this thesis, an approach for correctly handling the occlusion of virtual graphical objects by the anatomy of the patient was developed [10]. By solving this problem, it becomes possible for the user to easily determine whether a graphical object – like the rendering of a surgical tool or an instrument trajectory – is supposed to be in front of or behind the patient. A volumetric dataset containing the relevant part of the patient's anatomy is acquired before the surgical intervention using a medical scanning procedure, e.g., computed tomography or magnetic resonance imaging. After the initial registration, the IGS system requires that the patient's position and orientation remain fixed relative to the coordinate system of the infrared cameras. This means that the new method is a special case of static occlusion handling as described in Section 2.6.1.

The basic idea of static occlusing handling is to define a polygonal representation of real objects in the environment of the user. In most applications, relatively simple phantom models are used. Unlike such polygonal meshes of manageable size, models derived from medical volume datasets often consist of millions of triangles. This problem is aggravated by the steadily increasing image resolution provided by modern medical scanners. Therefore, in order to prevent occlusion handling from having an exceedingly negative impact on the overall frame rate, an application-specific method for reducing the number of triangles was devised for the new approach which is described here.

### Volume Preprocessing Pipeline

A volume preprocessing pipeline for reducing the polygonal complexity of models generated from medical datasets is the core of the new occlusion handling system. In contrast to general mesh simplification schemes, this method tries to preserve a highly detailed model of the outer surface of the anatomy while completely eliminating inner structures invisible from the outside. This is achieved by extracting the visual hull volume, a binary volume in which all parts of the volume dataset that cannot been seen from the outside (e.g., anatomical cavities) have full intensity. In order to compute this visual hull volume, a number of first-hit raycasting tests from six orthogonal directions is performed. This can be considered a simple approximative simulation of possible outside views of the volume dataset. Figure 2.37 illustrates the visual hull volume concept.

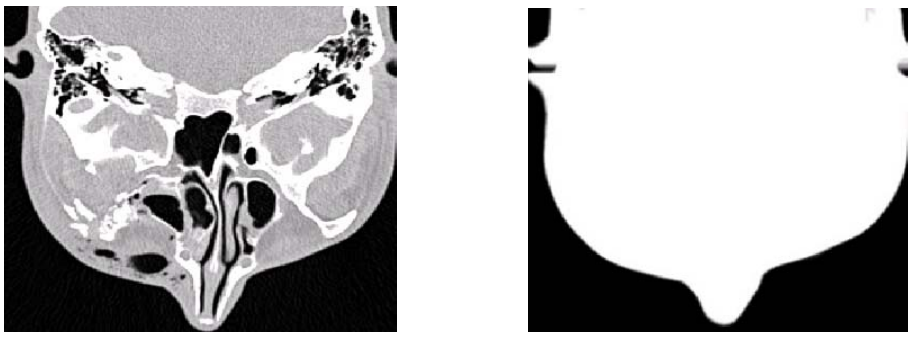

(a) Original volume (b) Visual hull volume

Figure 2.37: Comparison of an image slice in the input and visual hull volumes for a skull dataset.

### First-Hit Raycasting

The elementary operation of the visual hull algorithm is the casting of a single ray parallel to one of the coordinate axes into the volume dataset. The ray is iteratively traversed until a voxel intensity above a user-defined threshold is encountered. Until this threshold condition is met or the ray leaves the volume boundaries, zero values are written into a second volume dataset at the same coordinates. This resulting volume, the visual hull volume, is initialized with full voxel intensities (255 in case of 8-bit volumes) beforehand. The raycasting process can be considered an approximative simulation of the user viewing the outer surface of the volume dataset from one direction.

Parallel rays are generated over the entire area of each face of the bounding box of the volume. The visual hull volume is initialized with full intensities only at the very beginning of the process. Each of the raycasting iterations then works on the already modified volume. Thus the empty parts surrounding the actual anatomy are "carved out" of the volume consecutively. The principle of the consecutive first-hit raycasting iterations is described as pseudocode in Algorithm 1.

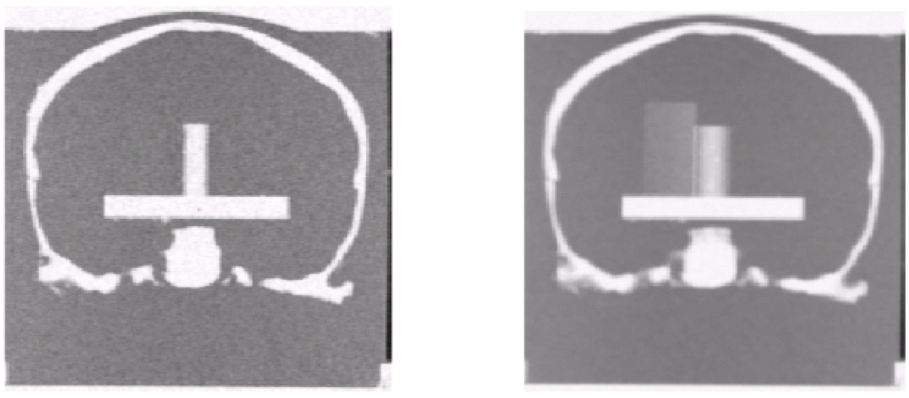
While the first-hit raycasting method works well for most volumetric datsets, ray iterations can be terminated erratically in volumes with an above-average level of noise. The raycasting process is therefore embedded into a volume processing pipeline comprising additional filtering steps.

**Algorithm 1** Computation of visual hull volume with iterative raycasting.

```
originalVolume:  VolumeData;
visualHull:      VolumeData;

rayCastingDirs:  Vector3d[6] :=
   {{0,0,1}, {0,0,-1}, {0,1,0}, {0,-1,0}, {1,0,0}, {-1,0,0}};

procedure preprocessVolume(threshold:  double)
  startPos, pos:  Vector3d;
begin
  initializeWithFullIntensity(visualHull);

  for directionLoop := 1 to 6 do
    for startPos ∈ all positions on corresponding
                   face of volume boundary do

      pos := startPos;

      while (pos within volume boundaries) do

        if (originalVolume[pos] > threshold)
           break;

        visualHull[pos] := 0;
        pos := pos + rayCastingDirs[directionLoop];

      done
    done
  done
end
```

### Additional Volume Filtering Steps

Before the actual raycasting operation is applied to the volume dataset, two filtering steps are performed. The first is a standard low-pass Gaussian volume filter. It reduces noise in the volume, but also tends to soften the edges of anatomical structures. Like for most steps in the volume processing pipeline, the user can opt against using the Gaussian filter and directly use the original volume as input for the next stage. If Gaussian smoothing is applied to the dataset, the standard deviation used by the filter can be selected by the user. After this low-pass filtering step, morphological operations are applied to the volume. These operations can remove isolated islands of low or high intensity (e.g., small holes) from the volume. In order to achieve this improvement, at first a 3D morphological opening and then a morphological closing operation are performed. (For a more detailed description of morphological operators see, e.g., [97].) Here again, the user can skip this preprocessing step and directly continue with the next stage. The kernel size used for the morphological opening and closing steps can be manually adjusted if necessary. Figure 2.38 shows the effect of the volume preprocessing stage on one slice of a volume with a relatively high level of noise.



(a) Input volume (b) Preprocessed volume

Figure 2.38: One slice of a volume dataset of the plastic skull before and after application of volume preprocessing steps (Gaussian filter and morphological operators).

The computation of the visual hull volume from the preprocessed volume dataset only yields voxels with either full intensity or zero intensity, as described above. Since such a binary volume dataset leads to visible artefacts in the subsequent polygonal iso-surface extraction, another volume filtering step is applied. In order to smooth hard edges and remove possible remaining noise, first a median filter and then another Gaussian smoothing are used. Unlike for previous steps of the pipeline, fixed kernel sizes have proven to work reliably for the postprocessing of visual hull volumes. The median filter uses a kernel size of 3x3x3 voxels, while the Gaussian postprocessing filter has a standard deviation of 1.5 voxels. Since the median filter step is not necessary for all volumes, it can be skipped. The result of postprocessing with a Gaussian filter operation is illustrated in Figure 2.39.

If areas of full voxel intensity are located at the boundary of the volume dataset, the subsequent iso-surface extraction is unable to generate a suitable polygonal representation. In such cases, the computed iso-surface will have holes at the respective places of the visual hull volume. Such holes make the polygonal surface unsuitable for the task of occlusion handling. The user can thus choose to apply a final postprocessing step to the volume, in which it is padded with a surrounding layer of zero intensity voxels. This additional layer causes the iso-surface

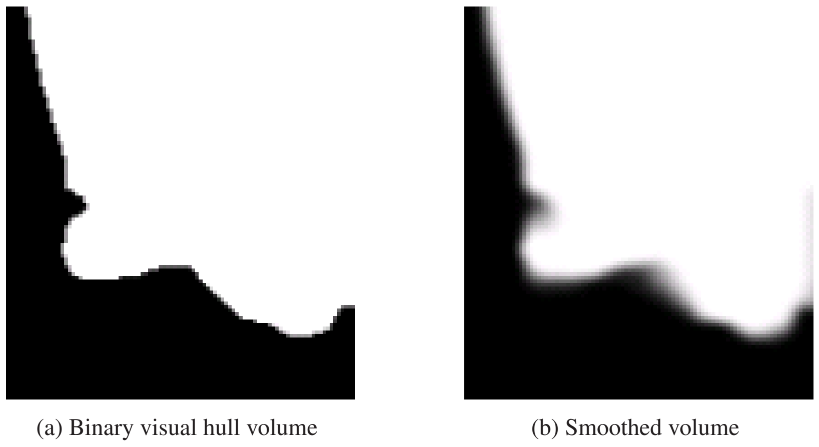


Figure 2.39: After the raycasting process, the visual hull volume is smoothed using a Gaussian filter. A magnified detail of one slice of a visual hull volume is shown.

extraction algorithm to generate a closed outside surface for the whole volume. Figure 2.40 gives an overview of the entire volume processing pipeline used for computing the visual hull volume. Several of the pipeline steps were implemented using respective functionality from the standard Visualization Toolkit (VTK) library [175].

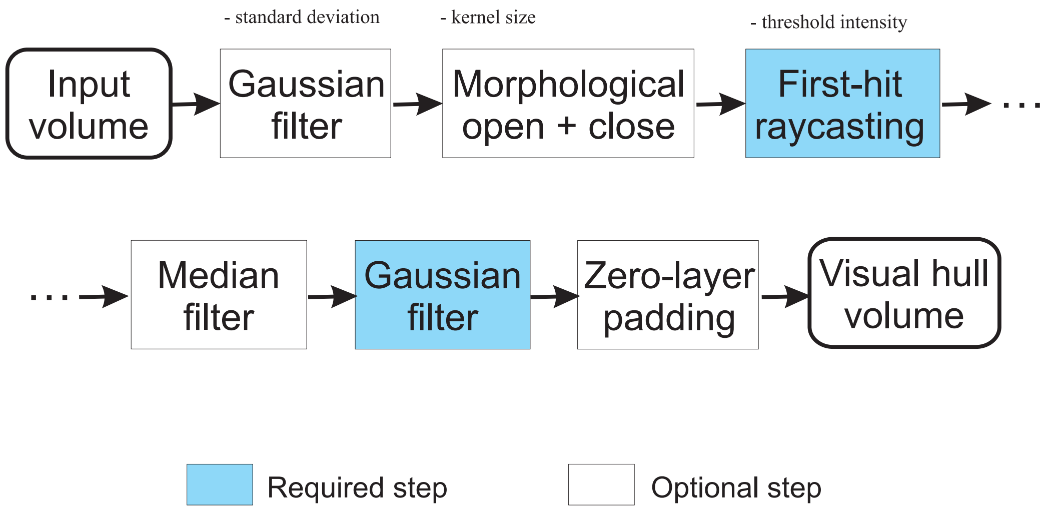


Figure 2.40: Overview of the volume processing pipeline for computing the visual hull volume. The steps in boxes with white backgrounds are optional and do not need to be applied in all cases. The items above some of the boxes list user-definable parameters for the respective processing steps.

### Integration into Rendering Process

A polygonal iso-surface is extracted from the visual hull volume using the marching cubes algorithm [137]. Since the visual hull volume contains a representation of the outer surface as the interface between full and zero intensity, the half of the full intensity is selected as iso-surface threshold (128 in case of 8-bit volumes).

This visual hull iso-surface is then used for suppressing the display of occluded graphical objects during the augmented reality image composition process. In order to achieve this effect, the iso-surface is rendered in a special way before the actual virtual objects. Writing to the color buffer is disabled while the visual hull surface is rendered, so that only the z-buffer (depth buffer) is altered. Then color buffer access is enabled again. When the graphical objects are rendered, their color values overwrite the original camera image, but pixels lying behind the (invisible) visual hull are suppressed. Since the graphical model contained in the volumetric dataset corresponds to actual patient anatomy, the impression of virtual objects being occluded by the patient is created. This modified rendering process is illustrated in Algorithm 2.

**Algorithm 2** Modified rendering process for occlusion handling, using OpenGL-like terminology.

```
glClear(GL_DEPTH_BUFFER_BIT);
drawCameraImage();
computeAndLoadTransformationMatrix();

// Disable color buffer access
glDrawBuffer(GL_NONE);
renderOcclusionVisualHullSurface();

// Enable color buffer access
glDrawBuffer(GL_BACK);
renderVirtualObjects();
```

Correct static occlusion handling in medical augmented reality requires an accurate patient registration. Since the fragments of the virtual objects in the AR environment are compared with the depth of graphical primitives of the visual hull iso-surface, it is necessary that this iso-surface is located and oriented so that it corresponds to the actual patient anatomy. As mentioned above, one of the advantages of the *ARGUS* system is the fact that it automatically benefits from the registration mechanisms of the IGS device (see Section 2.3.3). The *ARGUS* framework acquires the patient registration transformation from the IGS system. This transformation matrix is then used for correctly positioning and rotating the visual hull iso-surface. Therefore, the integration of the occlusion handling method into the rendering process can be realized with the straightforward approach shown in Algorithm 2.

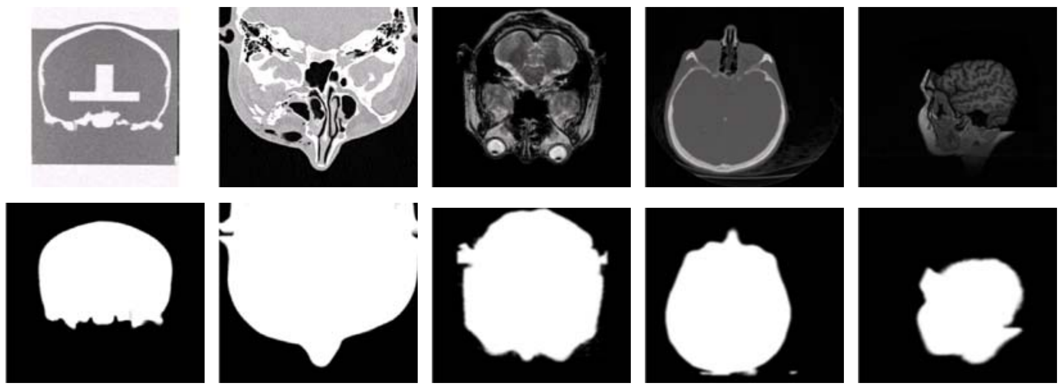
### 2.6.3 Occlusion Handling Results

The visual hull volume extraction was tested with several datasets. Figure 2.41 gives an overview of the five medical volumes used for the evaluation, and their dimensions are listed in Table 2.7. “Plastic skull” (Fig. 2.41a) is an MRI dataset of the plastic skull mock-up used in the actual augmented reality setup. Figures 2.41b and 2.41c show CT scans of the heads of patients who were treated at the department of maxillofacial surgery of the University Hospital Tübingen. The last two test datasets (Figures 2.41d and 2.41e) were taken from the Stanford volume data archive [134]. Each of the columns shows one slice from the input volume dataset and the corresponding slice in the computed visual hull volume. As illustrated in Figure 2.41, the visual hull volumes generated by the pipeline are an accurate representation of the outer surface of the patient’s anatomy. Assuming that sufficiently good parameters for thresholds and filter kernel sizes are selected by the user, almost all computed volume slices are free of errors. The visual hull volume slice in Figure 2.41d shows one rare exception, where artifacts at the lower end of the skull were not removed by the algorithm (see Fig. 2.42 for a detailed

Table 2.7: Dimensions and bit depth per voxel of the test datasets.

| Dataset | Size X | Size Y | Size Z | Depth |
|---|---|---|---|---|
| Plastic skull | 512 | 512 | 160 | 8 bits |
| Patient A | 512 | 512 | 147 | 8 bits |
| Patient B | 256 | 256 | 40 | 8 bits |
| CTHead | 256 | 256 | 113 | 8 bits |
| MRBrain | 256 | 256 | 109 | 8 bits |

illustration). The defect in this case was caused by a physical contact between the skull of the patient and the CT tube it was placed in, which deteriorates the CT scan. This artifact could have been prevented, however, if a better prepared volume dataset had been used as input. This could have been achieved by preprocessing it with appropriate segmentation or clipping methods.



(a) Plastic skull (b) Patient A (c) Patient B (d) CTHead (e) MRBrain

Figure 2.41: Visual hull volumes computed for five volume datasets. The first row shows one slice from each input volume, while the second row shows the corresponding slices from the visual hull volumes. The dataset in column 2.41a contains the plastic skull used in the actual AR system setup, columns 2.41b and 2.41c are from clinical cases at the University Hospital Tübingen, and 2.41d and 2.41e were taken from the Stanford volume data archive.

Figure 2.43 shows iso-surfaces generated from a medical dataset. The first image (Figure 2.43a) is a rendering of the iso-surface extracted directly from the the original input volume. In Figure 2.43b, the iso-surface of the corresponding visual hull volume is shown. As illustrated in the images, most of the detail of the surface of the anatomy is preserved by the volume processing pipeline. A slight smoothing of the surface occurs due to the Gaussian filters used in the visual hull extraction process. The quality of the computed visual hull iso-surface is more than sufficient for the occlusion handling task, especially since it mainly depends on the accuracy of the surface silhouette.

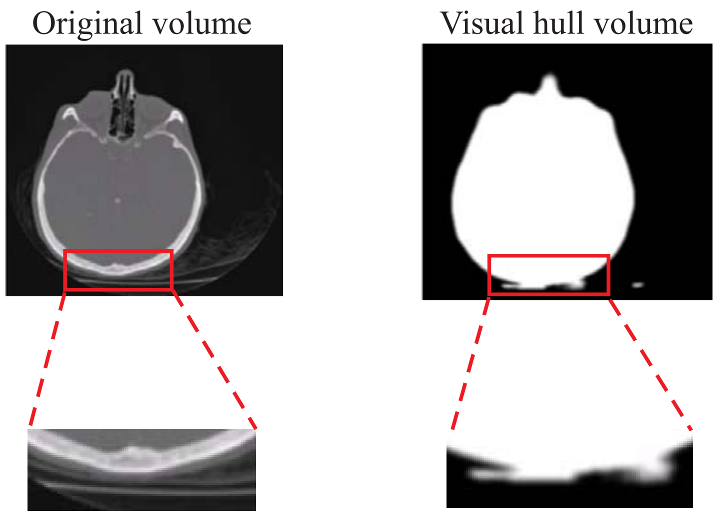
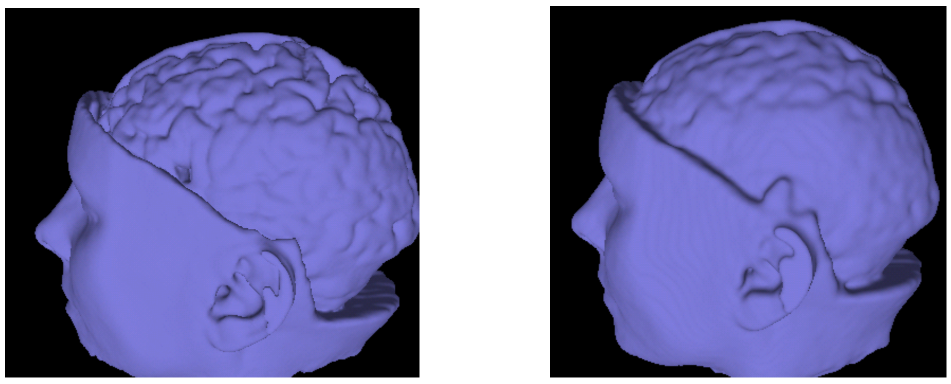


Figure 2.42: Artifacts in the visual hull volume of the “CTHead” dataset.



(a) Iso-surface from unprocessed volume (b) Iso-surface from visual hull volume

Figure 2.43: Comparison of the iso-surfaces generated from the unprocessed and visual hull volumes of the “MRBrain” dataset.

Iso-surfaces were extracted for both the original input and the visual hull volumes of all five test datasets. A comparison of the triangle counts of the generated surfaces is shown in Table 2.8. Threshold and filtering parameters in both cases were selected so that an outer surface suitable for occlusion handling was obtained. As demonstrated in the table, a triangle count reduction of at least one third is achieved in all but one cases. While reduction rates of close to or over 50% frequently occur, only dataset “CTHead” is not significantly simplified by the volume processing pipeline, due to the already low complexity of the input volume. The measured average triangle count reduction rate of 42.9% confirms the simplification performance of the algorithm.

The effect of the occlusion handling method on augmented images is illustrated in Figure 2.44. The images shown here are screenshots from interactive real-time sessions in the *ARGUS* framework with occlusion handling based on the plastic skull dataset. Figures 2.44a and 2.44b illustrate that the problem of unintelligible depth relationships is solved by the occlusion handling method. The situation in these images is very similar to the scenario presented in Figure 2.36, where the graphical representation of the tracked tool always appears to occlude

Table 2.8: Triangle counts of iso-surfaces generated from unprocessed input and visual hull volumes.

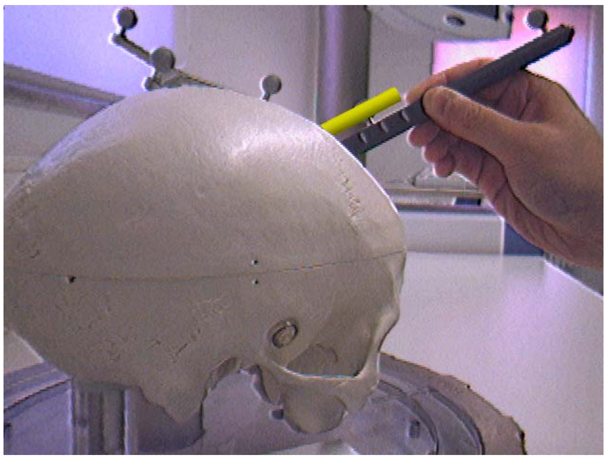
| **Dataset** | **Input** (triangle count) | **Visual hull** (triangle count) | **Reduction** |
|---|---|---|---|
| Plastic skull | 2,219K | 1,279K | 42.4% |
| Patient A | 3,312K | 1,627K | 50.9% |
| Patient B | 767K | 230K | 70.0% |
| CTHead | 339K | 283K | 16.5% |
| MRBrain | 433K | 282K | 34.9% |
| Average | | | 42.9% |

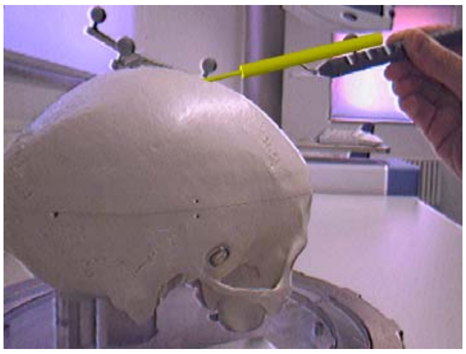
the plastic skull. With occlusion handling turned on, the surgical tool model is now correctly occluded whenever the actual tool is behind the real plastic skull, as demonstrated in Figures 2.44a and 2.44b. Most of the graphical representation of the tool cannot be seen because the rendering of the respective pixels is suppressed by the depth buffer values of the visual hull iso-surface. This creates the visual impression that the graphical model is actually occluded by the real skull itself. The simultaneous correct occlusion of multiple surgical tool models is shown in Figures 2.44c and 2.44d. In Figure 2.44c, only one of the two tools is behind the skull. Therefore, the second tool is completely visible. By contrast, both tools are located behing the plastic skull in Figure 2.44d, causing both graphical models to be partially occluded. The fact that small details in the medical dataset are preserved by the volume preprocessing pipeline is illustrated in Figure 2.44e. In this image, the surgical tool model is occluded by the visual hull iso-surface of the cheek-bone, while the geometry remains visible outside of it.

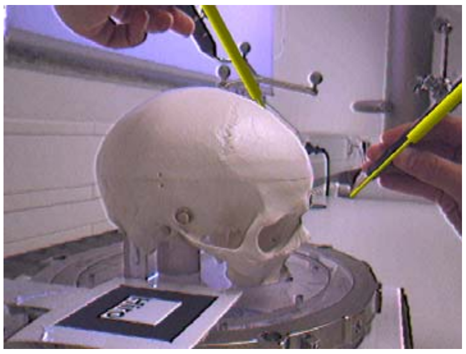
In order to examine the benefits of the visual hull volume extraction algorithm, the times required for rendering iso-surfaces generated from unprocessed and processed volumes were compared. Table 2.9 lists average rendering times measured during test sessions. Each test run was at least 60 seconds long, during which similar typical camera movements were performed. The average time required for rendering the extracted iso-surface into the depth buffer was calculated over all frames generated during the test run (typically more than 1000 frames). The tests were performed on a computer system with an Intel Pentium 4 Xeon processor running at 2.66 GHz, 2 GBytes of main memory and a graphics card based on an NVidia GeForce FX 5900 chipset. While some of the absolute speedups listed in Table 2.9 may not seem very large, the absolute reductions of about 20 milliseconds achieved for the datsets “Plastic skull” and “Patient A” are a significant improvement. The iso-surface rendering is embedded in a complex augmented reality pipeline of video image acquisition, download of tracking information over the network and the final rendering stage. Therefore, rendering times in the range of 40 to 50 milliseconds for the occlusion handling can seriously hamper real-time image generation in the AR system. This is confirmed by a perceivable drop of the overall frame rate when iso-surface rendering is enabled for a larger dataset without the use of visual hull volume extraction. As shown in Table 2.9, considerable performance improvements through the algorithm were generally measured in terms of relative speedup. This acceleration quality can be assumed to scale well for cases in which larger datasets or several datasets are used.

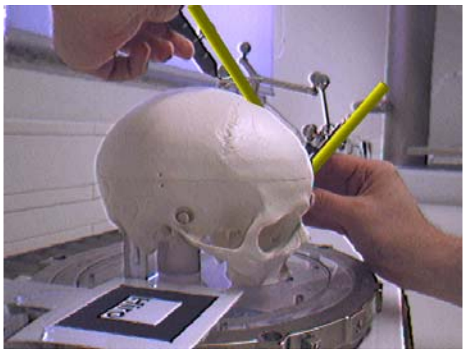
Table 2.9: Average time required for rendering iso-surfaces into the depth buffer.

| **Dataset** | **Input** (msecs) | **Visual hull** (msecs) | **Speedup** |
|---|---|---|---|
| Plastic skull | 35.7 | 17.4 | 105.2% |
| Patient A | 51.7 | 23.3 | 121.9% |
| Patient B | 14.4 | 6.4 | 125.0% |
| CTHead | 4.6 | 3.7 | 24.3% |
| MRBrain | 5.9 | 4.9 | 20.4% |
| Average | | | 79.4% |


(a) Single tracked tool occluded by plastic skull


(b) Single tracked tool occluded by plastic skull


(c) Two tracked tools, one is behind plastic skull


(d) Two tracked tools, both are behind plastic skull

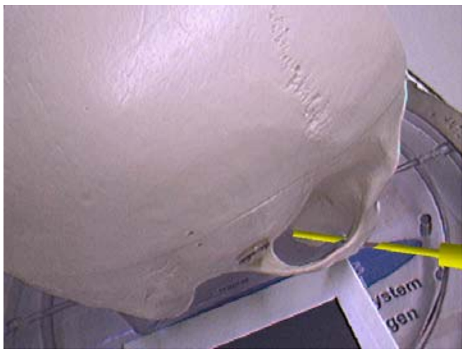

(e) Occlusion of tracked tool by cheek-bone

Figure 2.44: The effect of the occlusion handling method in the augmented reality setup. Graphical representations of surgical tools are used as virtual objects.

### 2.6.4 Discussion

The presented visual hull volume extraction process aims at ensuring that the occlusion handling does not have a negative impact on the frame rate. In order achieve this aim, an iso-surface is generated which contains only a highly detailed representation of the outer surface of the patient anatomy. All inner structures are removed since they are not required for the occlusion handling algorithm.

The described first-hit raycasting strategy with rays parallel to coordinate axes might not work optimally for certain volumetric objects. However, the method has proven to generate good results for all test datasets. Alternative approaches would include volume segmentation algorithms like region growing (see [97]). These, however, generally have a high computational complexity and often are not robust when used with non-watertight objects. The presented new approach typically only takes a few seconds to compute even for large datasets, and it practically never leaks through the outer surface of the anatomy. At the same time, the number of parameters to be selected by the user is relatively small.

The typical approach to reducing the number of triangles of an iso-surface is to apply a mesh decimation algorithm (e.g., as described in [176]). The major disadvantage of such methods in the context of occlusion handling is the fact that they do not distinguish between inner and outside surfaces. In order to achieve a similar reduction of the triangle count, a significant degradation of the quality of the outside surface geometry would have to be accepted. Still, a conservative mesh decimation algorithm could be considered as a postprocessing step for the visual hull surfaces generated by the presented volume processing pipeline. This could further reduce the number of triangles while maintaining a sufficient visual quality of the outside hull.

## 2.7 Summary

In this chapter, the basic design and extensions of the *ARGUS* framework for medical augmented reality were presented. Unlike other research systems, this solution is based on existing equipment for medical visualization and navigation. Devices for image guided surgery are becoming increasingly widespread and are found in many operating rooms. Therefore, the transition of augmented reality into the clinical practice can be facilitated by the fact that IGS systems are well tested, stable, and certified for medical settings. Since the VectorVision image guided surgery devices have been designed specifically for medicine, many practical problems like working in a sterile environment have already been taken care of. An IGS-based augmented reality system benefits from these qualities. Assuming that IGS equipment exists for a certain medical application, additionally only a standard computer system and webcam are necessary for building a basic AR system. The initialization of the image guided surgery system can be performed within a few minutes by a trained surgeon, while the one-time camera calibration step (Sec. 2.3.2) is only required in case of a configuration change.

The advanced hybrid tracking scheme described in Section 2.4 is capable of significantly improving the overlay accuracy in medical AR. While it has the typical limitations of a (partly) vision-based tracking method and is sensitive to the quality of the acquired camera image, it can typically deliver a much better pose estimation than the purely infrared-based system. Future research topics could include the development of a (semi)automatic model definition method for hybrid tracking and a better application-specific adaptation of the approach to the requirements of a medical setting.

The *ARGUS* framework is still an experimental system, and it has not yet been used in the clinical practice. Some additional challenges will have to be addressed before it can be applied intraoperatively. These tasks include the selection of an appropriate display device. Currently, the video streams generated by the AR application are displayed on a conventional computer monitor. Moreover, a comprehensive experimental study on the overlay accuracy of the system could be performed for confirming the reliability of the system.

In Section 2.5, a novel concept for untethered user interaction in medical augmented reality was presented. Unlike previously described interaction methods, this approach is based exclusively on certified image guided surgery equipment. By contrast, specialized 3D tracking or interaction devices are usually designed for applications in VR or engineering and are ill-suited for medicine. The interaction modes supported by the new method can be used for a wide range of applications. The menu system can be used for changing the parameters of an advanced information display like volume or multi-modal rendering. Even conventional functions like loading a patient dataset or initiating the patient registration procedure could be triggered using the novel menu system.

The *ARGUS* user interaction system presented in this thesis has also been used as the basis for a new method for semiautomatic volume classification. In this application, intuitive augmented reality interaction delivers the input data for a machine learning algorithm, which computes the volume rendering transfer function [5, 6, 128]. This development demonstrates that complex application cases can be realized based on the *ARGUS* framework and the associated user interaction library.

The occlusion handling method described in Section 2.6 contributes to the ongoing research on correctly rendering depth relationships in augmented reality. While the associated volume processing pipeline was designed for volumetric datasets containing patient anatomy, the approach could easily be employed for other applications in which volume datasets are available. These include fields like engineering or production planning. The tests have shown that the method is capable of generating easily understandable augmented images, in which virtual objects appear correctly occluded by the patient anatomy. This makes a better interpretation of the spatial relationships in the AR scene possible. The improved visual quality achieved through the new occlusion handling method could further promote the acceptance of augmented reality in medicine.

As discussed in the introduction of this thesis, the design of improved rendering methods for augmented reality is an important area of research (see Chapter 1). The presented occlusion handling system was developed as an advanced approach to more realistic image generation in AR. An understandable representation of the spatial relationships between objects in the augmented environment can contribute significantly to the usefulness of an AR system. In the following chapters of this thesis, the topic of advanced rendering methods and alternative display modes for augmented reality will be investigated further.

CHAPTER 3

# Stylized Augmented Reality

## 3.1 Introduction

As discussed in Section 1.2, conventional augmented reality systems generally use straightforward methods for overlaying graphical objects over the real environment. Video see-through augmented reality systems acquire the digital input video stream and display the current video frame as background image for the augmented view. The graphical primitives which constitute virtual objects in the AR scene are then rendered over the background image using standard computer graphics methods. Common real-time graphics libraries like OpenGL [181] or high-level frameworks based on them are often utilized for this task. The resulting renderings contain the typical artifacts of computer generated graphics, e.g., aliasing caused by the rasterization process. Figure 3.1 illustrates the standard method for overlaying graphical objects in conventional augmented reality.

Moreover, simplified lighting and shading models are normally used in real-time computer graphics. Common local illumination methods for the vertices of graphical primitives rely on manually placed virtual light sources and manually assigned material parameters. Simple interpolation methods like Gouraud shading then spread the computed brightness values over the graphical models [90]. Even if more sophisticated rendering methods with advanced illumination and shading are used, the problem of mismatched scene generation parameters still persists. Since light sources and material properties are defined manually during the definition of the AR scene, they do not correspond to the lighting conditions in the actual environment. This becomes particularly apparent when the camera is moved, which often results in varying image brightness and coloring, while the appearance of computer generated virtual objects remains invariable.

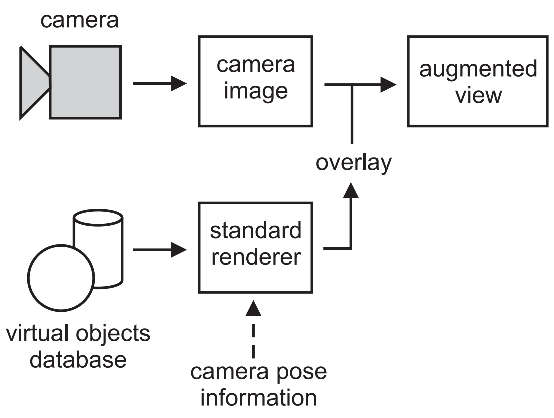
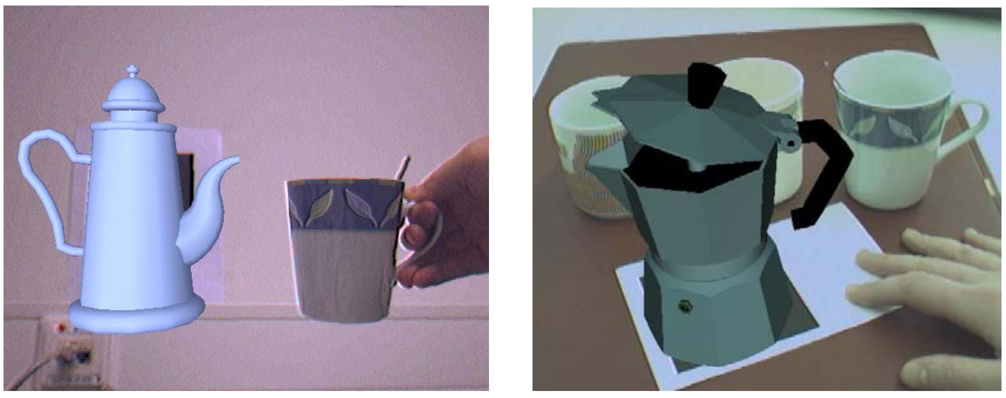


Figure 3.1: The image mixing process in conventional AR.

Due to the large discrepancies in visual appearance, even in a still image an observer can easily distinguish virtual objects from the real camera background in most cases. This is illustrated in the examples in Figure 3.2. It is obvious that the teapot and the coffee maker are virtual objects in the respective augmented scenes, since they appear to be "pasted over" the camera image.



(a) Virtual teapot in AR environment (b) Virtual coffee maker in AR environment

Figure 3.2: Two examples of virtual objects in conventional AR images. Due to the artificial look of the virtual objects generated by standard rendering methods, they can easily be distinguished from the real environment.

In this thesis, an alternative method for generating augmented video streams is proposed. This novel approach creates a stylized reproduction of the AR images. The basic idea of the new method for generating augmented images is that **the same type of artistic or illustrative stylization is applied to both the digital camera image and the graphical objects**. This way, adapted levels of realism are created in the real camera image and the virtual models that compose the AR video output. Since the same type of stylization is applied to both the real background image and the graphical models, they become significantly more difficult to distinguish. The resulting augmented output images create a novel experience for the user of an augmented reality system, and they possibly create a better feeling of immersion. This new approach to AR image generation has been named *stylized augmented reality*. It was proposed and presented *for the very first time* in the context of this thesis [15].

The principle of stylized augmented reality can basically be implemented with any type of artistic or illustrative rendering style. Two preconditions have to be fulfilled so that a useful stylized output video stream is generated:

1. **Visual similarity.** The stylization processes applied to the camera image and the virtual objects have to create similar-looking output images. This is necessary in order to ensure that real and graphical objects become less distinguishable.

2. **Real-time performance.** The complete stylized AR system has to be capable of generating an output video stream at real-time frame rates in order to make an application in interactive scenarios possible.

In the context of this thesis, three different kinds of stylization were applied to augmented video streams. In the original implementation of stylized AR, a cartoon-like look was applied to the AR images. Additionally, an artistic type of stylization was realized. In this rendering style, the output images are composed of brush strokes. The application of an illustrative rendering style to augmented reality is described in Chapter 4.

In the remainder of this chapter, Section 3.1.1 gives an overview of related work on rendering methods for augmented reality and on artistic and illustrative rendering. In Section 3.2, the original cartoon-like stylization algorithm for augmented video streams is described. Section 3.3 presents an advanced cartoon-like stylization method which works as a postprocessing filter on the GPU. The alternative, artistic rendering style for AR images is discussed in Section 3.4. Section 3.5 describes the design and results of a psychophysical study on the effectiveness of stylized augmented reality. Finally, Section 3.6 concludes this chapter with a summary.

### 3.1.1 Related Work

An approach that is complementary to the concept of applying stylization to AR images is the attempt to improve the realism of virtual objects. This approach is generally known as *photometric registration*. The photometric registration problem is defined as the task of adapting the illumination conditions and overall visual appearance of two images while preserving their original, unstylized look. This way, a better visual correlation between the virtual objects and the camera image can also be achieved. An early method for the correct automatic illumination of virtual objects added to real images was described by Debevec [75]. Research has also been done into methods of analyzing the real illumination conditions in an interactive augmented reality setup. Examples of this approach include the work of Gibson et al. on photometric reconstruction for mixed reality [95]. The system of Kanbara and Yokoya analyzes the distribution of real light sources, which is then used for adapting the representation of graphical objects [121]. Their method requires a special marker and mirror ball to be visible in the camera image in order to compute the environment light map. A similar technique which also utilizes an acquired environment illumination map is proposed by Agusanto et al. [41]. In their system, a mirror ball and special camera are used in a specific procedure for determining the lighting conditions in the scene beforehand. Heymann et al. have proposed a GPU-accelerated rendering technique for realistic illumination in AR based on information from a captured mirror sphere [112, 113].

An advanced type of photometric registration is the method developed by Okumura et al. for analyzing the blur in the camera image [155, 156]. Blur is produced if the camera optics is not correctly focused on the currently observed objects. Since no blur is present in the renderings of virtual objects, they look different from the camera image. Okumura et al. use the measured size of blur for adapting the appearance of virtual objects. This is achieved by applying a corresponding blur filter to the rendered image of the graphical scene elements.

Another method for making virtual objects appear more realistic is to add shadowing to the AR image. This creates the impression that shadows are cast from virtual objects on physical surfaces. Haller et al. describe an algorithm for displaying such shadows in augmented reality [103, 104]. A similar technique is used for a user study on the effects of shadowing in AR by Sugano et al. [190]. As a drawback of these methods, a model of the geometry of objects in the real world is required. This model needs to be generated beforehand and is assumed to remain static.

The concept of stylized augmented reality is based on artistic and illustrative image generation and filtering. Artistic and illustrative rendering and image processing have been areas of very active research for many years. Strothotte and Schlechtweg have published a good survey of methods used in the field [189]. Another overview of various artistic and illustrative techniques is given by Gooch and Gooch [99]. Hertzmann and Perlin have presented a method for the stylization of video streams using an artistic painterly style [110]. One example of an algorithm for the cartoon-like stylization of photographs is the work presented by DeCarlo and Santella [76]. Their technique uses a combination of color segmentation and edge detection, which partly inspired the algorithms for cartoon-like stylized AR presented in Sections 3.2 and 3.3. However, their method requires several minutes for processing an input image. An algorithm for the semi-automatic conversion of a real video sequence into a cartoon-like video has been presented by Wang et al. [202]. This method produces results of good visual quality, but it is an offline algorithm and computationally too expensive for real-time applications. Moreover, a certain amount of user interaction is required for the specification of semantic regions in some video frames.

Some researchers have integrated artistic and illustrative rendering styles into virtual and augmented environments. The application of a specific artistic rendering method in virtual reality was presented by Klein et al. [126]. In this system, a completely virtual scene is displayed using an artistic style, and no video information is included. The first method which integrated painterly rendering into augmented reality was presented by Haller and Sperl [103, 106]. However, they applied artistic rendering techniques only to the virtual objects, whereas the camera image was displayed in its original, unprocessed form. After the initial publication of the concept of stylized augmented reality in [15], Haller et al. developed a method for displaying both the camera image and virtual objects in AR in a "loose and sketchy" style [105].

It could be argued that stylized augmented reality is a variation of the original definition of augmented reality. While the camera image is only used as an unprocessed backdrop in conventional AR, the image of the real environment is significantly modified in stylized AR. Mann has introduced the term *mediated reality* for systems which alter the view of the real environment [142]. Later, different applications of mediated reality were described [101, 102, 192]. The concept of stylized AR can therefore be considered a form of mediated reality.

## 3.2 Cartoon-like Stylization of Augmented Reality Images

In this section, the original algorithm for generating stylized augmented video streams, which was published in [15], is described. This approach handles the digital camera image and the graphical objects separately. However, the design of these two branches of the image generation pipeline aims at generating a similar, cartoon-like style for the real and the virtual scene elements in the output image. In this system, the camera image is processed before it is used as background in the image mixing process. An image stylization filter is applied to the input camera image. The aim of this filtering step is to create a simplified, stylized version of the current camera view. After the camera image has been processed, the virtual objects are rendered over it. However, unlike in conventional AR, a stylized rendering scheme is used instead of standard methods. A special renderer creates a cartoon-like representation of the graphical objects. It is based on the cartoon-like rendering method described by Lander [130, 131]. An overview of the entire process is given in Figure 3.3.

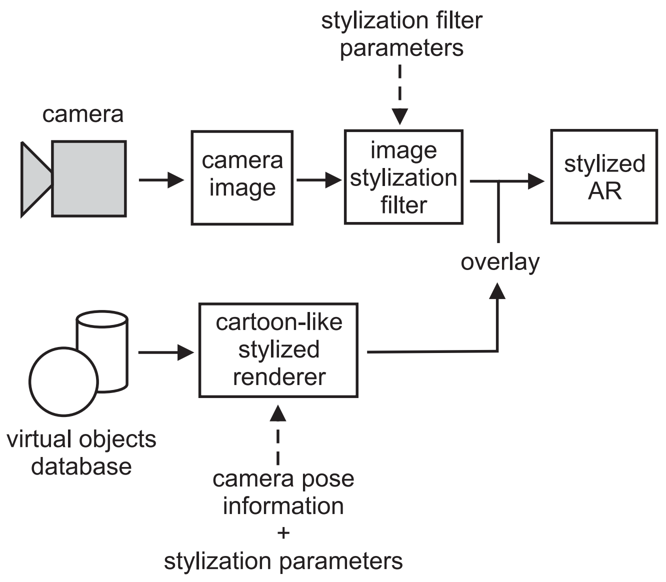
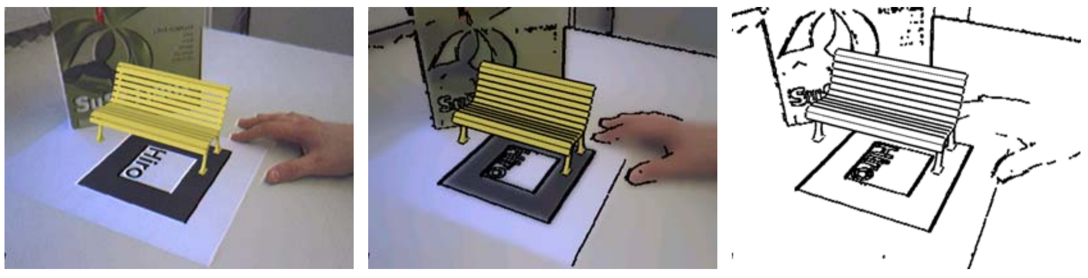

stylization filter parameters
camera
camera image
image stylization filter
stylized AR
overlay
cartoon-like stylized renderer
virtual objects database
camera pose information + stylization parameters

Figure 3.3: The image mixing process used for cartoon-like stylized augmented reality.

As pointed out in Section 3.1, an important precondition for a useful system of stylized augmented reality is the ability to generate images in real-time. Therefore, a combination of camera image filter and stylized rendering method was developed which is fast enough to ensure high overall frame rates. It is important to note that both the image filter and the rendering component can be customized using a set of parameters. In order to obtain a similar type of stylization for both AR image layers, the parameters for the filter and rendering components must be adjusted accordingly. The algorithm described in this section is capable of generating two slightly different subtypes of stylization. *Cartoon-like* AR images consist of flat, uniformly colored patches enclosed by silhouette lines. In a second, *sketch-like* mode, only black silhouettes are visible in front of a white background. Figure 3.4 shows a comparison of the two stylization subtypes.



(a) Conventional AR (b) Cartoon-like stylization (c) Sketch-like stylization

Figure 3.4: Comparison of cartoon-like and sketch-like stylization subtypes for a simple AR scene containing the virtual model of a bench.

### 3.2.1 Stylization Filter for the Camera Image

The image filter used by the original cartoon-like stylization method is designed to simplify and stylize the input camera image. In order to achieve this effect, two separate steps are performed. The first step aims at reducing detail in the camera image by generating large, uniformly colored regions based on the original image information. The second stage detects high-contrast edges in the image. A postprocessing step is applied to the edge image in order to generate thick lines which are adequate for the desired cartoon-like look.

Subsequently, the images generated by the two steps of the stylization filter are combined. The edge image is a binary map, in which detected edges have full intensity. Since the silhouette edges in the final output image are supposed to be black, the edge image is inverted and then combined with the color image using the binary AND operation.

If the sketch-like stylization mode is used, the color segmentation step is skipped. In this case, only edge detection is performed, and the inverted edge map is the output of the stylization filter. This way an image with a white background and black silhouette lines is created. An overview of the image filtering process is shown in Figure 3.5.

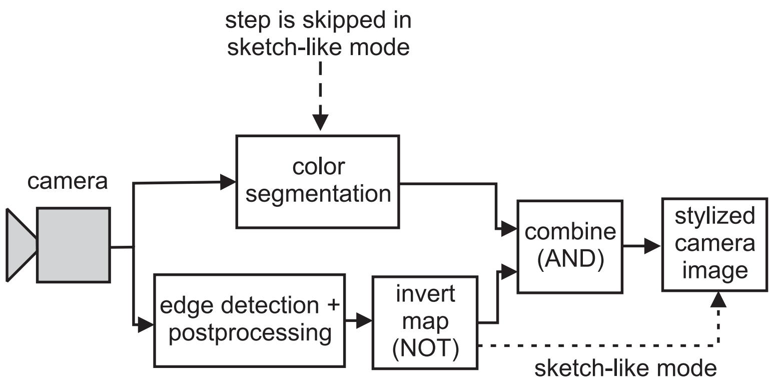


Figure 3.5: Overview of the image stylization filter used in the original cartoon-like stylized AR system.

An essential design goal for the elements of the camera image filter is to achieve an interactive execution speed. Since the filter is embedded in a complex augmented reality and stylization pipeline, it must not take more than a few milliseconds to compute. Due to these strict runtime requirements, most of the filtering process was designed from scratch and specifically adapted for the purpose of stylized augmented reality. In order to achieve a sufficiently short execution time for the algorithm, the OpenCV library [116] was used for speed optimized image processing on the CPU.

### Color Segmentation

The first stage of the stylization filter aims at converting the input image into an image consisting of regions which are mostly uniformly colored. This way a simplified and stylized "coloring book" look is created. This task is closely related to color segmentation. However, due to the time constraints it is not possible to achieve a true full segmentation in this filtering step. The algorithm used in the filtering process is based on bilateral image filtering, which was described by Tomasi and Manduchi [194]. The basic idea of bilateral filtering is to take spatial distance as well as signal difference into account when smoothing a function. Unlike a Gaussian filter, the bilateral filter therefore leaves high-contrast edges mostly unaltered, while it has a strong smoothing effect on homogeneous regions.

Given the multi-channel (RGB) image function $\mathbf{f}$, the bilateral filter computes the smoothed image $\mathbf{h}$ using the following equation:

$$\mathbf{h}(\mathbf{x}) = k^{-1}(\mathbf{x}) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathbf{f}(\xi) c(\xi, \mathbf{x}) s(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x}))\, d\xi \tag{3.1}$$

In Equation 3.1, $\mathbf{x}$ is the currently regarded point in the output image, and the integral is computed in two dimensions over neighboring image points $\xi$. In the discrete case, this is equivalent to a weighted sum of image pixels $\mathbf{f}(\xi)$ in the neighborhood of $\mathbf{x}$. The weight is a product of two factors. $c(\xi, \mathbf{x})$ is a function of the vector difference $\xi - \mathbf{x}$, i.e., the spatial distance. The second factor, $s(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x}))$ depends on the similarity of values in the color channels, $\mathbf{f}(\xi) - \mathbf{f}(\mathbf{x})$. In the implementation used by the cartoon-like image filtering algorithm, both $c$ and $s$ are Gaussian functions:

$$\begin{aligned} c(\xi, \mathbf{x}) &= e^{-\frac{1}{2}\left(\frac{|\xi - \mathbf{x}|}{\sigma_d}\right)^2} &(3.2)\\ s(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x})) &= e^{-\frac{1}{2}\left(\frac{|\mathbf{f}(\xi) - \mathbf{f}(\mathbf{x})|}{\sigma_r}\right)^2} &(3.3) \end{aligned}$$

Note that the value of $c$ is a function of the Euclidean distance between $\xi$ and $\mathbf{x}$ (Equation 3.2), while $s$ depends on the absolute value of the color difference $\mathbf{f}(\xi) - \mathbf{f}(\mathbf{x})$ (Equation 3.3). The standard deviations of the Gaussian functions, $\sigma_d$ and $\sigma_r$, determine the properties of the smoothing and can be chosen by the user as parameters for the stylization filter. In order to maintain the overall brightness of the image, the integral is divided by the normalization factor $k(\mathbf{x})$, which is computed as shown in Equation 3.4.

$$k(\mathbf{x}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\xi, \mathbf{x}) s(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x}))\, d\xi \tag{3.4}$$

The effect of the bilateral filter is that smoothing only occurs in places where nearby pixels have similar colors. In such places in the image, both $c(\xi, \mathbf{x})$ and $s(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x}))$ are large. If near the currently regarded pixel colors are present which are relatively far away in color space, smoothing is suppressed. Thus strong edges in the image are preserved.

The disadvantage of bilateral filtering for stylized AR is that it is computationally too expensive. Due to the necessity for computing the vector differences, the Gaussian functions, and the normalization factor, even an optimized implementation is too slow for the real-time requirements. Therefore, a Gaussian pyramid of iteratively shrunk images is created, and the bilateral filter is only applied to the smallest version. The use of resolution pyramids for image processing is described for instance in [46].

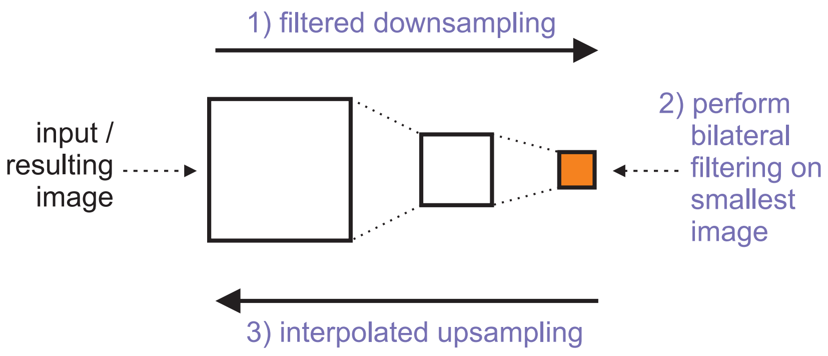
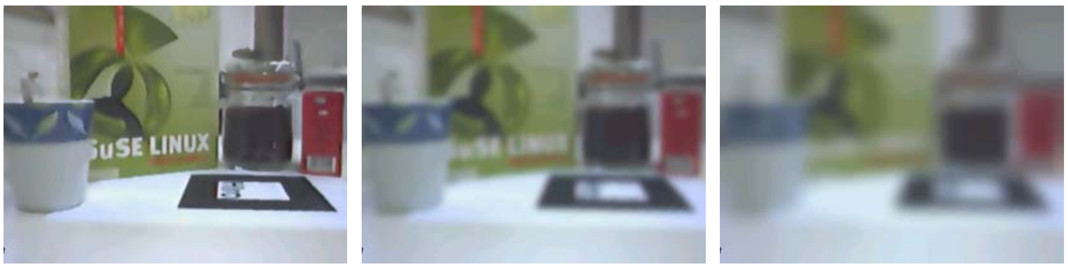

1) filtered downsampling
input / resulting image
2) perform bilateral filtering on smallest image
3) interpolated upsampling

Figure 3.6: Construction of Gaussian pyramid for speeding up the bilateral filter.

Figure 3.6 illustrates the principle of using a Gaussian pyramid for speeding up the color segmentation step. For each level of the pyramid, the image resolution is reduced in both dimensions by a factor of two. This results in an image with only a quarter of the number of pixels of the previous level. The image is filtered with a standard Gaussian filter before the subsampling takes place, reducing aliasing effects. A bilateral filter with standard deviations $\sigma_d$ and $\sigma_r$ is then applied to the top level of the pyramid, the smallest version of the image. The bilateral filter is repeatedly applied to the image in order to achieve a better color segmentation result. The number of iterations can be selected by the user. Afterwards, the filtered image is iteratively scaled up, each time by a factor of two in both dimensions. In this process, a modified Gaussian filter is used for a smooth interpolation of missing pixels. Finally, an image with the same resolution as the original camera image is created, which is the output of the color segmentation step. The number of levels of the Gaussian pyramid can be selected by the user as a parameter for the stylized AR system.



(a) One pyramid level (b) Three pyramid levels (c) Four pyramid levels

Figure 3.7: Color segmentation results using a different number of Gaussian pyramid levels for the same scene.

The required computation time for the bilateral filter decreases with an increasing number of pyramid levels. On the other hand, the resulting image becomes more blurred if more pyramid levels are used. In Table 3.1, measured runtimes for the entire color segmentation process, including construction and decomposition of the Gaussian pyramid, are listed (column

Table 3.1: Measured execution times of the color segmentation process.

| #Pyramid levels | Filter time | Frames/sec. |
|---|---|---|
| 1 | 0.217s | 4.57 fps |
| 2 | 0.092s | 10.79 fps |
| 3 | 0.063s | 16.00 fps |
| 4 | 0.055s | 18.00 fps |

"filter time"). These benchmarks for the color segmentation process, which is executed entirely on the CPU, were measured on a computer system with an Intel Pentium 4 Xeon processor running at 2.66 GHz. The third column of the table contains average measured overall frame rates of the AR system, if only color segmentation is used for processing the camera image. The table clearly shows that a higher number of pyramid levels significantly increases overall system performance. Figure 3.7 shows three color segmented camera images of the same scene and illustrates that a greater number of pyramid levels leads to a more blurred result.

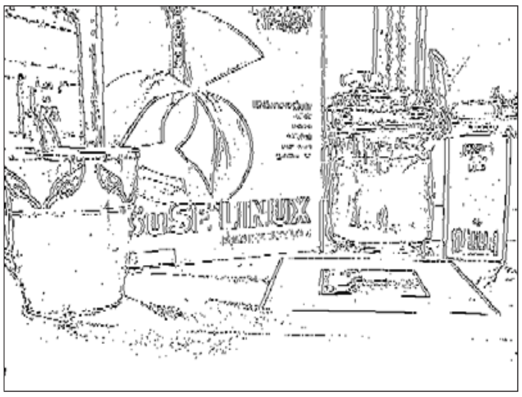
### Edge Detection

In the second stage of the image stylization filter, edges are detected in the camera image. The detected edges are used as black silhouette outlines in the stylized camera image, resulting in the desired "cartoon-like" or "sketch-like" look. The Canny edge detector [69] is used in the cartoon-like stylization system. Note that the edge detection step is performed on the original camera image, not on the result of the color segmentation process.

The Canny edge detector is based on a standard gradient computation, normally using the Sobel operator, followed by methods for suppressing erreneous responses. The final step of the Canny detector is the so-called hysteresis, which eliminates erreneous detection responses based on two threshold values. The larger threshold is used for finding definite edge pixels, which can be thought of as starting points for continuous edge segments. Any pixel adjacent to an already identified edge pixel is also marked as an edge pixel, if its gradient is greater than the smaller threshold. This way continuous edges are generated, while detached and weak gradient operator responses caused by noise are suppressed.

In the cartoon-like stylized AR system, these two thresholds for the Canny detector, $t_1$ and $t_2$, can be selected by the user. In order to achieve the goal of a stylized look with an emphasis on strong edges, rather large threshold values were found to be practical. Smaller thresholds can result in visually irrelevant edges being emphasized, or can even leave camera noise in the edge image. This is illustrated in Figure 3.8.

It is one of the properties of the Canny edge detector that the edge image it delivers consists of very thin edge segments (see Fig. 3.8b). For the purpose of the camera image filter in cartoon-like stylized AR, the black lines derived from the edge image should be much thicker. In order to achieve this effect, a morphological operator is applied to the image. The morphological dilation operation processes a binary image by assigning a value of one to all pixels in the neighborhood of any pixel which has a value of one [97]. As shown in Figure 3.9, this morphological postprocessing makes the edges appear thicker and removes discontinuities.


(a) Small thresholds result in image noise ($t_1$=80, $t_2$=60)

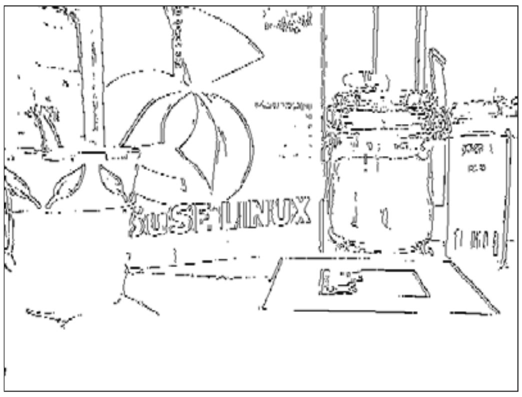

(b) Larger thresholds generate a better result ($t_1$=150, $t_2$=105)

Figure 3.8: Canny edge detection results for different thresholds.

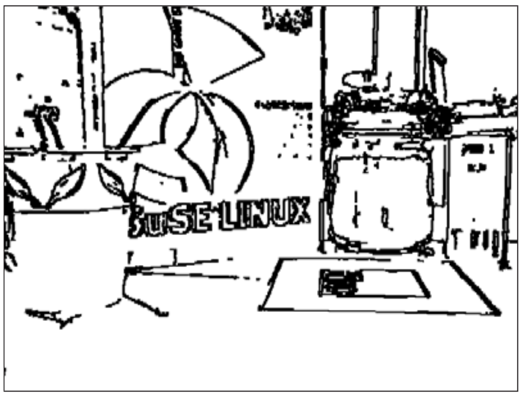


Figure 3.9: Application of morphological dilation to the edge image. The thresholds used for this image are the same as in Fig. 3.8b.

### 3.2.2 Cartoon-like Rendering Method for Virtual Objects

For rendering virtual objects in the cartoon-like stylized AR system, a stylized rendering method consisting of two main components is used. A technique for drawing thick silhouette outlines is applied to the geometry of the objects. Moreover, a custom shading method is utilized, which reduces the number of different pixel intensities in the generated image. These two methods make the rendered objects appear visually similar to the stylized camera image, which is used as background plane in the image mixing process.

#### Silhouette Rendering

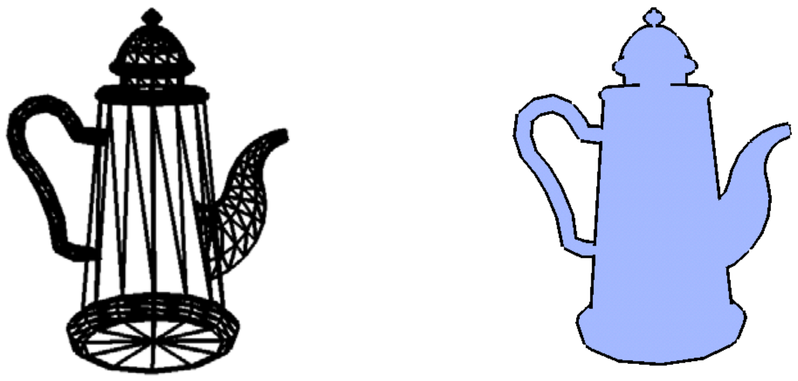
A simple method for rendering the silhouette outline of a graphical object by utilizing standard OpenGL functionality has been described by Lander [131]. This approach is based on drawing the geometry of each graphical object twice. In a first rendering pass, filled polygons are used for drawing the object. The Z-buffer test is applied in the normal way, so that pixels are only rendered if they are closer to the viewer than primitives which have already been drawn (depth test function `GL_LESS`). In this step, only front-facing polygons are rendered.

For the second rendering pass, the face culling performed by OpenGL is adjusted so that only back-facing polygons are drawn. These polygons, which face away from the camera, are rendered in wireframe mode. This means that only the outlines are drawn for each polyon. The wireframe representation is made up of thick lines. In this rendering pass, the Z-buffer is set up to allow the display of all pixels which are closer to the viewer or at the same depth as previously rendered pixels (depth test function `GL_LEQUAL`). This second rendering pass affects all the places in the image where back-facing and front-facing polygon edges have the same distance to the viewer. For most graphical objects, these places correspond to silhouette edges. The result of this rendering pass is that thick lines are visible where the silhouette outline of an object should be. The code fragment in Algorithm 3 explains how the two rendering passes can be performed using standard OpenGL calls.

**Algorithm 3** The two rendering passes used for silhouette rendering (taken from [131]).

```
glPolygonMode(GL_FRONT, GL_FILL);
glDepthFunc(GL_LESS);
glCullFace(GL_BACK);
DrawModel(); // Draw primitives of virtual object

glPolygonMode(GL_BACK, GL_LINE);
glDepthFunc(GL_LEQUAL);
glCullFace(GL_FRONT);
DrawModel(); // Draw primitives of virtual object
```

Figure 3.10 illustrates the effect of the silhouette rendering method. In the cartoon-like stylized AR system, the thickness of the silhouette lines can be selected by the user. Although this silhouette rendering approach is based on two-pass rendering, its impact on the overall system frame rate is negligible except for extremely large virtual models. The advantage of this method is that it does not require advanced OpenGL functionality (e.g., shaders) and is therefore highly portable.



(a) The second rendering pass: Only the outlines of back-facing polygons are drawn

(b) Final result of the silhouette rendering method

Figure 3.10: The silhouette rendered for the graphical model of a teapot.

### Non-linear Shading

The second method utilized by the stylized AR system for giving a cartoon-like look to virtual objects is non-linear shading. Normally, brightness and color parameters computed for the vertices of an object are spread over the visible surface using OpenGL shading. This means that either flat shading is applied, which uses constant color and brightness, or Gouraud shading, which uses linear interpolation between the vertex parameters over the area of a polygon. While flat shading is too simplistic even for the purposes of a stylized AR renderer, the application of Gouraud shading generates too many different intensity levels. Although Gouraud shading can make surfaces appear round (as seen on the teapot in Fig. 3.2a), it also creates a too artificial look that does not correspond well to the stylized background image.

The cartoon-like non-linear shading method, which was also described by Lander [130], only generates a limited and well-defined set of intensities. This is achieved by modulating the base color of the object with a special one-dimensional texture. The one-dimensional texture contains the representation of a function which translates texture coordinates to a brightness value in discrete, quantized steps. An example of a one-dimensional shading texture is shown in Figure 3.11. In the cartoon-like stylized AR system, the shading texture is generated automatically. The number of quantization steps and the base intensity can be selected by the user. In order to be able to represent a full light intensity, the shading texture always contains the full texel intensity in the last texels (127 for OpenGL signed byte intensity textures). In the implemenation of the stylized AR system, the cartoon-like renderer always uses a texture with a size of 32 texels, unlike the 16-texel texture used for clarity in Figure 3.11.

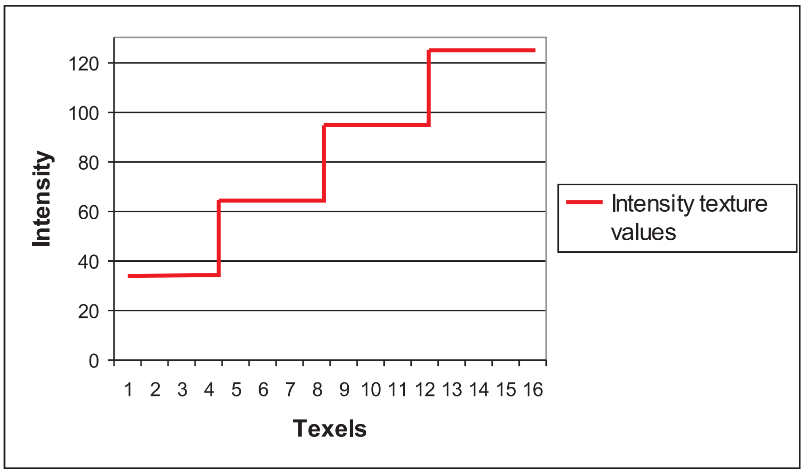


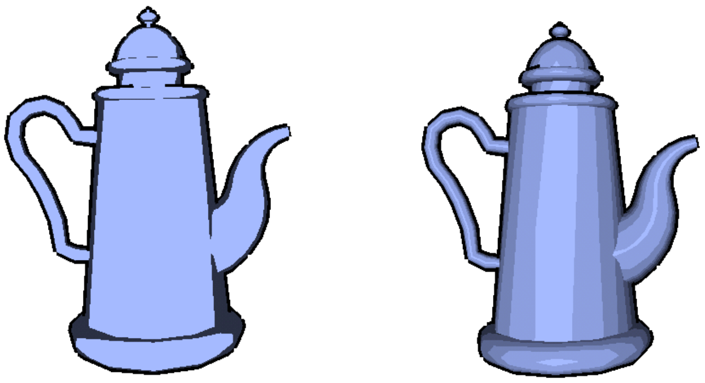
Figure 3.11: Example of a one-dimensional intensity texture.

For the actual rendering process, OpenGL lighting computations are disabled. Instead, light intensity is computed by the cartoon-like renderer for every vertex. This is done using simple diffuse reflection:

$$I_i = \mathbf{n_i} \cdot \mathbf{L} \tag{3.5}$$

In Equation 3.5, the brightness $I_i$ of vertex $i$ is computed as the dot product of its normal vector $\mathbf{n_i}$ and the light direction $\mathbf{L}$. In order to obtain a correct lighting, each vertex normal is rotated by the rotational component of the current OpenGL transformation matrix beforehand.

This ensures that the current viewing parameters are taken into account. The system maintains normalized $\mathbf{n_i}$ and $\mathbf{L}$, therefore the result of the dot product is always between -1 and 1. Negative values are clamped to zero. The resulting value between 0 and 1 is then used as index into the one-dimensional shading texture, i.e., as texture coordinate. This way the texture acts as a lookup table for the non-linear shading function. The effect of non-linear shading with a different number of quantization steps is illustrated in Figure 3.12. The cartoon-like stylized AR system provides functionality for manually specifying the light direction used by the shading algorithm.



(a) 2 quantization levels (b) 6 quantization levels

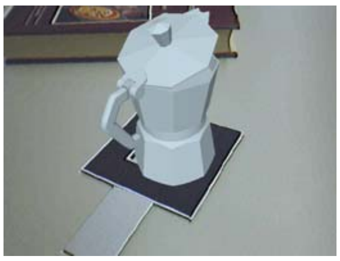
Figure 3.12: Non-linear shading with a different number of quantization levels in the shading texture.

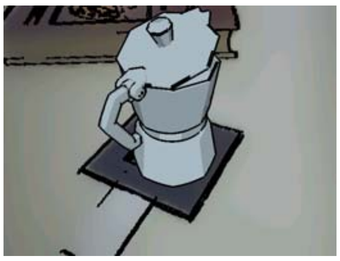
If the sketch-like stylization subtype (see Fig. 3.4c) is selected by the user, a special configuration is used for the virtual objects renderer. In this case, the front-facing polygons are rendered uniformly white, which is achieved by setting the entire one-dimensional shading texture to full intensity and choosing white as the base color of the object. The silhouette rendering process is performed as described in Section 3.2.2 with black selected as the outline color. This creates a pure black-and-white look for the virtual objects, which corresponds to the filtered camera image in the sketch-like style.

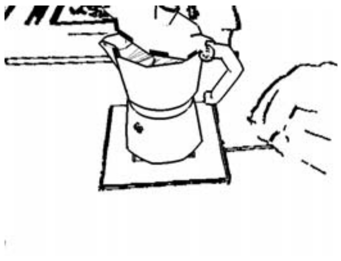
### 3.2.3 Results

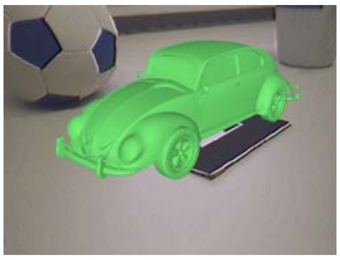
The described cartoon-like stylized augmented reality approach was tested with numerous different test scenes. The implementation of the stylized AR system is based on the *ARGUS* framework introduced in Chapter 2. The software contains an editor which is capable of importing 3D models in the standard Wavefront OBJ file format (see [148] for a format description). The user can freely place, scale and rotate the model. Moreover, a graphical user interface for adjusting all parameters of the camera image stylization filter and the stylized virtual objects renderer is provided. These include the number of Gaussian pyramid levels and standard deviations for the bilateral filter, as well as object color, lighting parameters, line thickness, and line color for the stylized rendering algorithm. Additionally, the user can choose between the cartoon-like and sketch-like stylization subtypes.

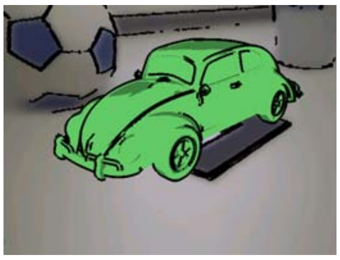
For all of the test scenes shown here, optical marker tracking based on the ARToolKit library was used [123]. The user-defined transformation of the 3D model is in relation to an ARToolKit marker, which determines the origin of the coordinate system.

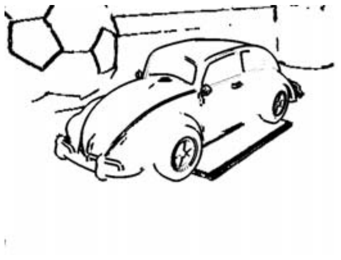

(a) Moka Express, conventional AR

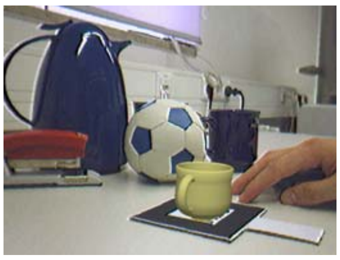

(b) Moka Express, cartoon-like stylization

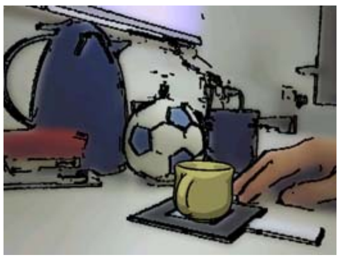

(c) Moka Express, sketch-like stylization

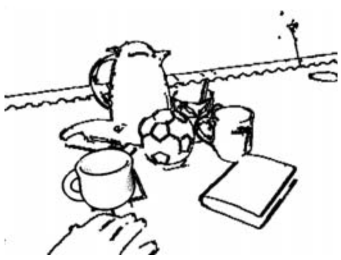

(d) VW Beetle, conventional AR

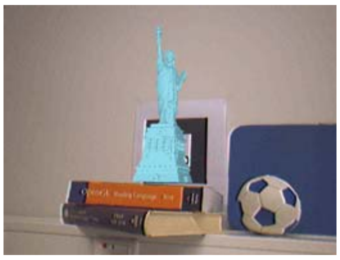

(e) VW Beetle, cartoon-like stylization

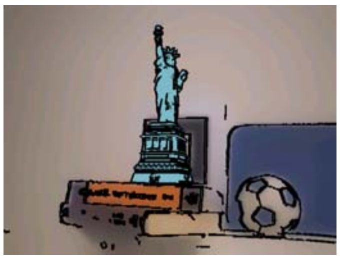

(f) VW Beetle, sketch-like stylization


(g) Teacup, conventional AR


(h) Teacup, cartoon-like stylization


(i) Teacup, sketch-like stylization


(j) Statue of Liberty, conventional AR


(k) Statue of Liberty, cartoon-like style

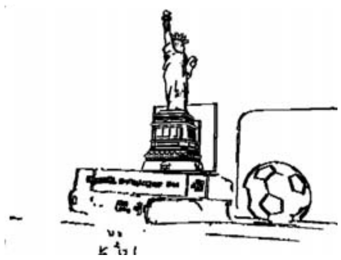
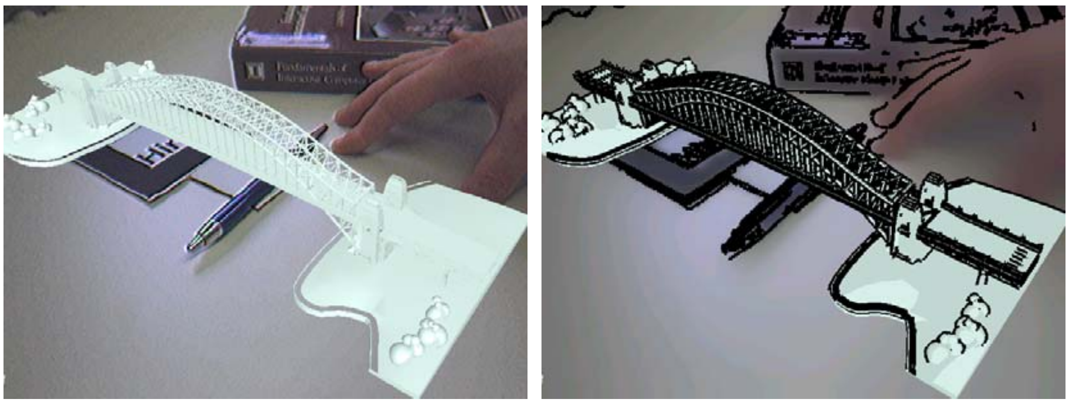

(l) Statue of Liberty, sketch-like style

Figure 3.13: Four example scenes illustrating the effect of the original cartoon-like stylized augmented reality approach. In each row, the leftmost column shows conventional AR rendering. The second and third column contain the stylized versions. Note that Figure 3.13c and 3.13i show the respective scene from a different angle.

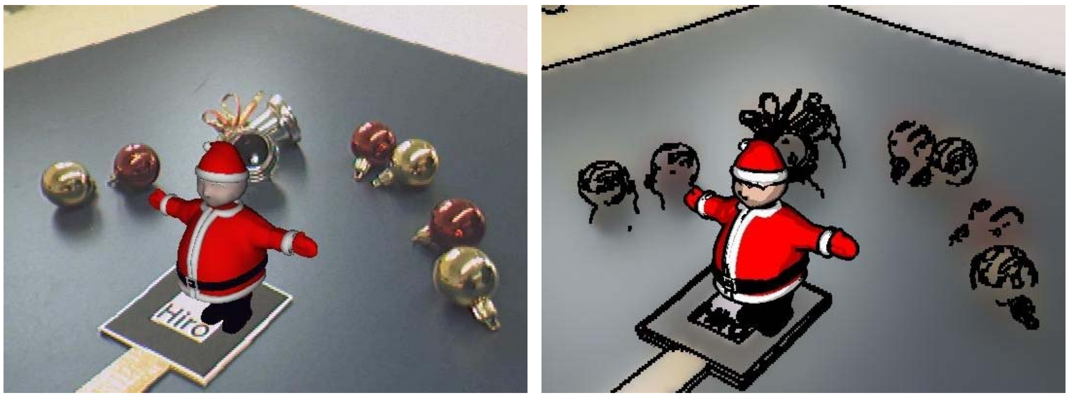
Figure 3.13 shows four augmented reality test scenes. In each row of images, the leftmost column contains the augmented scene as generated by conventional AR image composition. In the second column, the image is rendered with the presented cartoon-like style. The sketch-like stylization subtype for each test scene is depicted in the third column. Figures 3.14, 3.15 and 3.16 show three more example scenes demonstrating the original cartoon-like stylization algorithm.



(a) Bridge scene, conventional AR

(b) Bridge scene, cartoon-like stylization

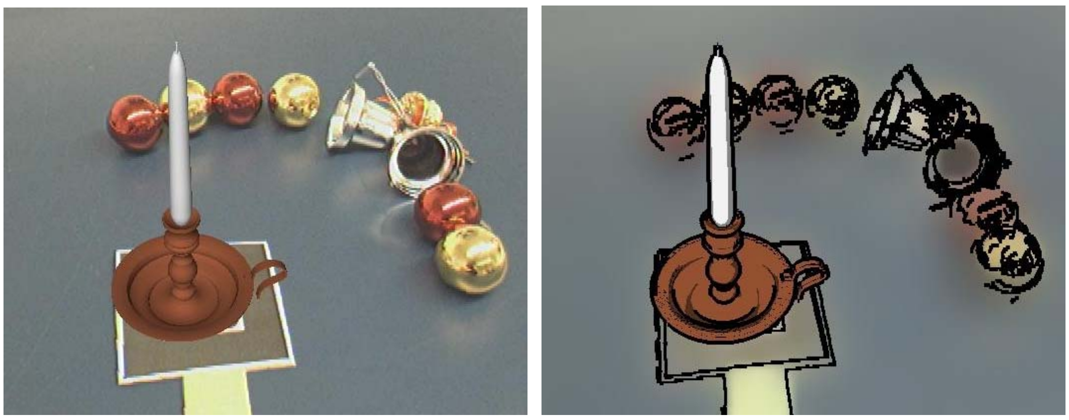
Figure 3.14: Example scene showing a virtual bridge model.



(a) Santa Claus model, conventional AR

(b) Santa Claus model, cartoon-like stylization

Figure 3.15: Santa Claus example scene for the original cartoon-like stylization method.



(a) Candle model, conventional AR

(b) Candle model, cartoon-like stylization

Figure 3.16: Candle example scene for the original cartoon-like stylization method.

It is the goal of the described cartoon-like stylization approach as well as the basic idea of stylized AR to achieve an improved immersion. This means that it becomes less obvious for the user whether an object in the augmented image is real or virtual. The example images in the Figures 3.13 - 3.16 show that this effect is achieved by the presented method. Especially for scenes in which the scale of the virtual object matches the physical world, the barrier between virtual and real is reduced. The coffee maker (Fig. 3.13b) and teacup (Fig. 3.13h) are good examples for scenes in which the virtual model appears to be a natural part of the real environment thanks to the stylized display method.

Several performance measurements of the described original cartoon-like stylized AR approach are listed in Table 3.2. These performance measurements were taken on a computer system with an Intel Pentium 4 Xeon processor running at 2.66 GHz and a graphics card based on an nVidia GeForce FX 6800GT chipset. Measurements were taken for two different virtual graphical models. The "teacup" dataset contains more than 81k vertices, while the "teapot" object is made up of only approximately 4k vertices. For each of the two models, interactive stylized AR sessions with a duration of at least 300 frames were examined. The benchmarks were performed for both the cartoon-like and the sketch-like stylization subtypes, with the virtual objects being visible in the generated output images during the entire experimental sessions. In Table 3.2, the third column lists the average overall frame rate measured in the test scenarios. The fourth column ("image filter") contains the measured average duration of the algorithm phase consisting of camera image acquisition and the image stylization filter (in milliseconds). The last column ("rendering") shows the average time required for rendering the virtual object, also listed in milliseconds.

Table 3.2: Benchmarks measured with the original cartoon-like stylized augmented reality system.

| **Scene** | **#Vertices** | **Frame rate** | **Image filter** | **Rendering** |
|---|---|---|---|---|
| | | (fps) | (msecs) | (msecs) |
| Teacup (cartoon style) | 81.7k | 8.4 | 64 | 55 |
| Teacup (sketch style) | 81.7k | 10.16 | 43 | 55 |
| Teapot (cartoon style) | 4.4k | 14.88 | 64 | 3 |
| Teapot (sketch style) | 4.4k | 21.49 | 43 | 3 |

The benchmark results show that the performance of the cartoon-like stylized AR system depends on several main factors. One is the complexity of the graphical model used in the augmented reality scene. Due to the specialized two-pass rendering approach (see Sec. 3.2.2), a higher polygon count leads to a significantly decreased rendering speed. Table 3.2 demonstrates that, as a second factor, the sketch-like stylization subtype is faster to generate than the cartoon-like style. The sketch-like stylization only requires the edge detection step of the stylization filter without any color simplification (see Sec. 3.2.1). Because the computationally expensive color simplification step can be skipped, the camera image filter takes significantly less time in the sketch-like mode (43 milliseconds) than for cartoon-like images (64 milliseconds). As additional factors, which have not been separately examined here, the number of Gaussian image pyramid levels and the number of bilateral filter iterations also influence the attainable frame rate. In all the benchmark scenarios listed in Table 3.2, two image pyramid levels and two bilteral filter iterations were used. A detailed discussion of the connection between the number of Gaussian pyramid levels and algorithm performance was shown in Table 3.1 (see Sec. 3.2.1).

The application of the original cartoon-like stylized AR system to various example scenarios has lead to the empirical observation that it is typically capable of delivering interactive frame rates. With the exception of AR scenes containing geometrically complex virtual models (like the “teacup” model in Table 3.2), frame rates of approximately 15 fps are normally achieved with typical algorithm parameters for the cartoon-like style.

## 3.3 Stylization of AR Video Streams on the GPU

The original cartoon-like stylization approach for augmented reality images discussed in Section 3.2 was the first implementation of the principle of stylized AR. It demonstrated the feasibility of a system which combines a specialized camera image filter and a cartoon-like renderer and generates an output video stream at interactive frame rates. Moreover, the original cartoon-like algorithm renders stylized augmented reality images in which real and virtual objects appear less distinguishable from each other. This way, the objective of stylized AR, to create an output video stream in which real and virtual scene elements are indistinguishable, is at least partially achieved.

This original stylized augmented reality approach, however, has some significant drawbacks. Its first problem is the limited graphical quality of the generated output. Especially in the processed camera image, strong flickering often occurs. This flickering is caused by the direct application of the line detection algorithm to the digital input image, which contains a large amount of noise. The rendered graphical objects can also contain visible artifacts due to the limited accuracy of the Z-Buffer, which leads to problems during the two-pass silhouette rendering step. Finally, although the original stylized AR system is capable of generating interactive frame rates, an even higher image generation speed is often desired for real-time performance. True real-time systems are usually considered to run at more than 20 fps (see [47]), resulting in a better experience and immersion for the user.

Therefore, a second, more advanced cartoon-like stylized AR system was developed in the context of this thesis [14, 38]. This advanced cartoon-like stylization algorithm solves many of the problems of the original approach. Although the output images generated by the newer system may look similar to the original cartoon-like stylization at first glance, the design of the advanced algorithm and its implementation are completely different. The advanced approach delivers output images of a higher visual quality. In particular, it eliminates flickering silhouette edges in the output video stream. The rendering performance of this newer algorithm is also significantly better than that of the original apprach. It can generate frame rates of 25 fps or more in typical scenarios.

In contrast to the original approach, the advanced cartoon-like stylization algorithm is executed entirely on the graphics processing unit. For each frame, a standard augmented reality pipeline first generates an output image containing the unaltered camera image with overlaid virtual objects. This original AR frame is rendered using standard rasterization methods on the graphics hardware and resides in its local frame buffer memory. A postprocessing filter is then applied to it, which is computed on the graphics processing unit (GPU). An overview of the approach is shown in Figure 3.17. Due to the design of the system, all image processing operations, which are computationally expensive, are performed efficiently by the dedicated graphics hardware. The image filtering steps are implemented as a set of specialized vertex and fragment shaders, utilizing the programmability of modern graphics processing units.

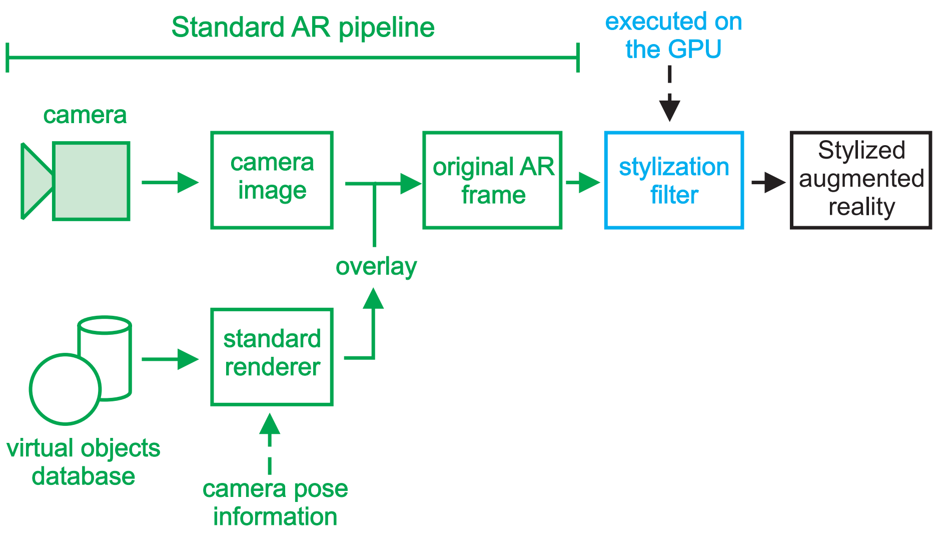


Figure 3.17: Overview of the advanced stylized augmented reality pipeline designed as a postprocessing filter.

Since the new method is based on the principle of applying a postprocessing filter to the original AR frame, it is related to image space stylization of acquired camera images. The cartoon-like stylization of single photographs (e.g., DeCarlo and Santella [76]) and entire video sequences (e.g., Wang et al. [202]) was described before. However, previously published approaches are usually offline algorithms and too slow for integration into a real-time image generation pipeline. Many existing algorithms also rely to some degree on user input for guiding the stylization procedure, which is not practical in an augmented reality scenario. The postprocessing step described here is based on common basic image operations like bilateral filtering and edge detection. However, these basic techniques were adapted and refined for the specific requirements of stylized augmented reality (see the enumeration in Section 3.1).

In the remainder of this section, Section 3.3.1 will give an overview of the algorithm. Subsequently, the color simplification step (Sec. 3.3.2) and the edge detection method (Sec. 3.3.3) are discussed in detail. Section 3.3.4 describes some aspects of the implementation of the algorithm. Experimental results obtained with the advanced cartoon-like stylization system are presented in Section 3.3.5.

### 3.3.1 Algorithm Overview

The cartoon-like postprocessing filter consists of two steps. In the first step, a simplified color image is computed from the original AR frame. The simplified color image is made up of mostly uniformly colored regions. A non-linear filter using a photometric weighting of pixels is the basis for this computation. The photometric filter is applied to a shrunk version of the input image. This way, a better color simplification is achieved, and the required computation time is reduced. Several filtering iterations are consecutively applied to the image. The repetition of the filter operation is necessary in order to achieve a sufficiently good color simplification. Figure 3.18 illustrates this procedure.

The second stage of the image stylization filter is an edge detection step. The simplified color image is the primary input for this operation. This way, the generated silhouette lines

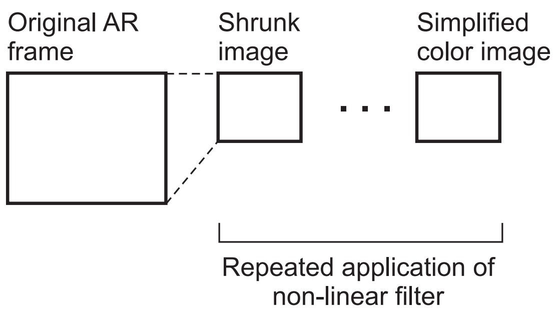


Figure 3.18: The simplified color image is generated with several iterations of a non-linear filter.

are located between similarly colored regions in the image, which is a good approximation of a cartoon-like rendering style. To a lesser degree, edges detected in the original AR frame are also taken into account when drawing the silhouette lines. The higher resolution of the original image compared to the shrunk color image can contribute some additional detail to the edge detection result. Figure 3.19 shows an overview of this filter stage.

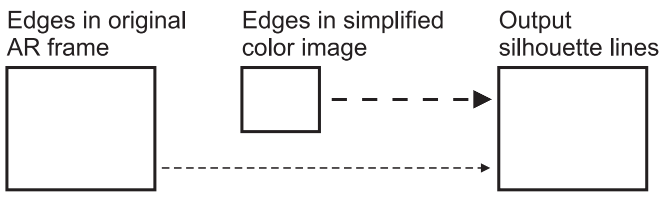


Figure 3.19: Edge detection results from the original image and the shrunk color image are combined.

In typical setups, most of the input for the edge detection step is taken from the simplified color image. It consists of mostly uniformly colored regions generated by the photometric filter. Therefore, edges detected in the simplified color image typically correspond quite well to the outer boundaries of physical or virtual objects.

Finally, the simplified color image is combined with the edge detection results. The color image is enlarged to the size of the original input image. The combined responses of the edge detection filters are drawn over the enlarged image as black lines. A specific weight function is used for computing a transparency for the detected edge pixels, which produces a smooth blending over the color image.

### 3.3.2 Generation of Simplified Color Image

At the beginning of the filtering process, a shrunk version of the original AR frame is rendered into the local frame buffer of the graphics card. This is done by drawing a rectangle textured with the original image. The texturing process is configured so that a smoothly scaled version of the image is produced. User-definable parameters, $shrunkImageWidth$ and $shrunkImageHeight$, specify the dimensions of the new image. The non-linear filter is then applied iteratively by using the output image of the last iteration as input texture for the next filtering step.

The non-linear filter used in the advanced cartoon-like stylization algorithm is inspired by bilateral filtering [194]. However, in contrast to the original cartoon-like approach (see Sec. 3.2.1) the filtering method is modified in order to achieve a better simplification result. The original bilateral filter algorithm combines geometric and photometric weights when adding up pixels in the neighborhood of the currently regarded pixel. While the geometric factor gives a greater weight to pixels closer to the current location, the photometric weight suppresses the influence of pixels with very dissimilar color values. In the context of cartoon-like stylization, however, it was empirically found that the photometric weight is sufficient for generating a useful simplification. Ignoring the geometric weight simplifies the algorithm and reduces the computational complexity. Moreover, this simplified non-linear filter produces very good visual results.

In addition to disregarding the geometric weight, the filter is also modified such that the photometric weight only depends on the actual color of each pixel. Each pixel is converted into the YUV color space before the filter is applied. In the YUV color space, the Y component represents the brightness of a pixel, while U and V are the chrominance (color) components [163]. For computing the weight of each pixel in the neighborhood, the new non-linear filter only takes the U and V components into account.

Again, the original RGB image function is denoted as $\mathbf{f}$, and the corresponding color coordinates in YUV space are called $\mathbf{f}_{UV}$. The non-linear filter computes the simplified RGB image $\mathbf{h}$ using the following equation:

$$\mathbf{h}(\mathbf{x}) = k^{-1}(\mathbf{x}) \sum_{\xi \in \Omega_{\mathbf{x}}} \mathbf{f}(\xi)\, s(\mathbf{f}_{UV}(\xi), \mathbf{f}_{UV}(\mathbf{x})) \tag{3.6}$$

In Equation 3.6, $\mathbf{x}$ is the currently regarded point in the output image. A weighted sum is computed over image points $\xi$ in the neighborhood $\Omega_{\mathbf{x}}$ of $\mathbf{x}$ in the input image. A quadratic image area is used as neighborhood for the summation. The weight $s(\mathbf{f}_{UV}(\xi), \mathbf{f}_{UV}(\mathbf{x}))$ depends on the similarity of values in the color channels, $\mathbf{f}_{UV}(\xi) - \mathbf{f}_{UV}(\mathbf{x})$. As in the original cartoon-like approach, $s$ is a Gaussian function:

$$s(\mathbf{f}_{UV}(\xi), \mathbf{f}_{UV}(\mathbf{x})) \;=\; e^{-\frac{1}{2}\left(\frac{|\mathbf{f}_{UV}(\xi)-\mathbf{f}_{UV}(\mathbf{x})|}{\sigma_p}\right)^2} \tag{3.7}$$

Note that $s$ is a function of the absolute value of the color difference, $\mathbf{f}_{UV}(\xi) - \mathbf{f}_{UV}(\mathbf{x})$ (Equation 3.7). The standard deviation $\sigma_p$ of the Gaussian function determines the properties of the color simplification and can be chosen by the user as a parameter for the algorithm. In order to maintain the overall brightness of the image, the weighted sum is divided by the normalization factor $k(\mathbf{x})$, which is computed as shown in Equation 3.8.

$$k(\mathbf{x}) = \sum_{\xi \in \Omega_{\mathbf{x}}} s(\mathbf{f}_{UV}(\xi), \mathbf{f}_{UV}(\mathbf{x})) \tag{3.8}$$

The effect of this non-linear filter is that an averaging of pixels only occurs in places where nearby pixels have similar colors. In such places in the image, $s(\mathbf{f}_{UV}(\xi), \mathbf{f}_{UV}(\mathbf{x}))$ is large. If near the currently regarded pixel colors are present which are far away in color space, they are not taken into account. Thus strong edges in the image are preserved.

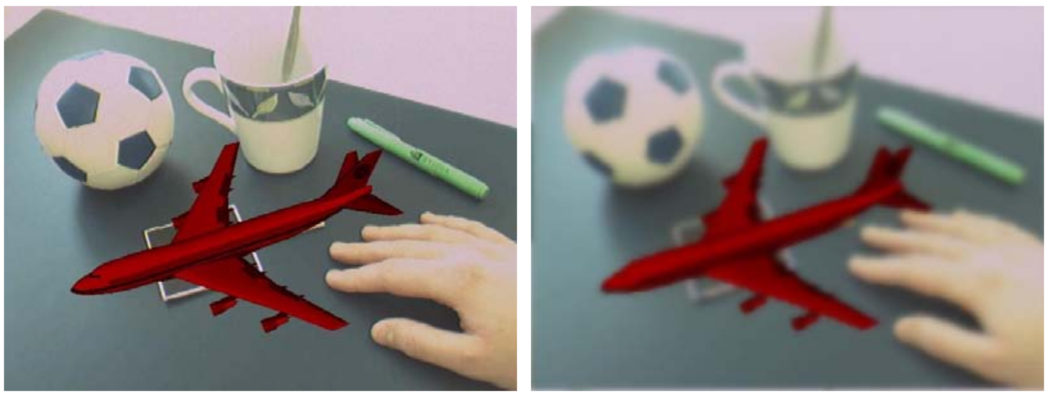

(a) Original AR frame (b) Simplified color image

Figure 3.20: Generation of the simplified color image for an original AR frame. In the augmented reality scene, a virtual plane model is overlaid over the camera image. (Parameters: *shrunkImageWidth*=240, *shrunkImageHeight*=180, $\sigma_p$=0.025, *numFilterSteps*=7)

In the implementation of the GPU-based cartoon-like stylization method, a small local neighborhood of 5 x 5 pixels is used for the weighted summation. As described above, the non-linear filter is applied several times. For each filtering step, the resulting image from the previous iteration is used as input. The number of color simplification iterations performed by the algorithm, $numFilterSteps$, can be chosen by the user. Figure 3.20 shows an example of a simplified color image computed for an original augmented reality frame.

The reason why it is possible to ignore the geometric weight in the computation of the non-linear filter is the fact that only a small pixel neighborhood is regarded. The main purpose of the geometric weight in the original bilateral filter is to reduce the influence of pixels which are father away in image space. Since only a small local neighborhood is taken into account in the implementation of the non-linear filter used in the advanced cartoon-like stylized AR system, the geometric weight is not necessary for this purpose. In this sense, the original bilateral filtering method is the combination of a Gaussian filter with the photometric weight, while the approach described here is the combination of a box filter with the photometric weight. To approximate a larger standard deviation used for the geometric weight in the original bilateral filtering algorithm, the number of filtering iterations can be increased. This increased number of iterations also leads to a enlarged area of influence for each pixel, resulting in a more blurred output image.

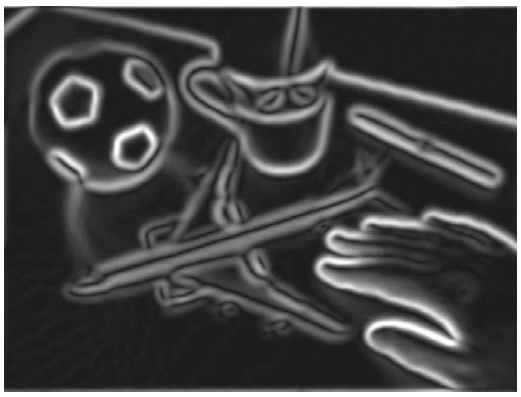
### 3.3.3 Adaptive Edge Detection based on Intensity and Color Contrasts

After the simplified color image has been generated, the edge detection step is performed. The algorithm uses the the Sobel edge detection filter for computing the partial derivatives of color channel values along the x-axis and the y-axis [97]. (The Canny algorithm is not used here because as an iterative process it would add several rendering passes to the filtering pipeline.) Here again, the pixels are converted into the YUV color space before the edge detection step. The image function of the simplified color image is denoted as $\mathbf{S}$, consisting of the channels $(S_Y, S_U, S_V)$. Correspondingly, the original AR frame $\mathbf{A}$ contains the YUV channels $(A_Y, A_U, A_V)$.

For each of the color channels of both images, two partial derivatives are calculated. In the case of the Y component, these are the derivatives $\frac{\partial S_Y}{\partial x}$, $\frac{\partial S_Y}{\partial y}$, $\frac{\partial A_Y}{\partial x}$, and $\frac{\partial A_Y}{\partial y}$. The U and V color channels are processed accordingly. Based on the partial derivatives, gradient magnitudes $(|\nabla S_Y|, |\nabla S_U|, |\nabla S_V|)$ are computed for the simplified color image, and $(|\nabla A_Y|, |\nabla A_U|, |\nabla A_V|)$ for the original AR frame.

An edge detection response is then calculated for each pixel using the gradient magnitudes. This response value is obtained through the weighted averaging of the local contrast in the intensity (Y) and color (U,V) channels. The relative weight of the intensity and color contrasts is determined by the parameter $\alpha \in [0;1]$. Equation 3.9 shows the computation of the edge detection response for the simplified color image, $edge_{(S)}$, and the original AR frame, $edge_{(A)}$. Using this method, the edge detection process can generate responses in locations with homogeneous intensities where the color channel gradient is large. The user can emphasize intensity contrasts or color contrasts for locating silhouette edges by adjusting the value of $\alpha$. Edge detection responses computed for the example AR scene in Figure 3.20 are shown in Figure 3.21.

$$\begin{aligned} edge_{(S)} &= (1-\alpha) \cdot |\nabla S_Y| + \alpha \cdot \frac{|\nabla S_U| + |\nabla S_V|}{2} \\ edge_{(A)} &= (1-\alpha) \cdot |\nabla A_Y| + \alpha \cdot \frac{|\nabla A_U| + |\nabla A_V|}{2} \end{aligned} \tag{3.9}$$


(a) $edge_{(S)}$

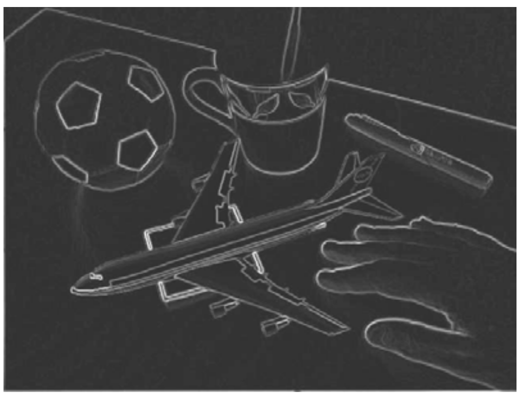

(b) $edge_{(A)}$

Figure 3.21: Edge detection responses for the original AR frame and simplified color image shown in Figure 3.20. (Parameter $\alpha$=0.3. Images have been brightened for better visibility.)

The two edge detection responses are then combined for determining the final silhouette intensity in the full-resolution output image. For every pixel position $(x_o, y_o)$ in the output image, the corresponding coordinates in the simplified color image are denoted as $(x_s, y_s)$. The parameter $\beta$ introduced in Equation 3.10 specifies the relative influence of the simplified color image and the original AR frame for silhouette detection. Note that the output coordinates $(x_o, y_o)$ are also used for accessing the edge detection responses of the original AR frame because it has an identical image resolution.

$$\begin{aligned} I_o(x_o, y_o) = (1-\beta) &\cdot smoothstep_{s0}^{s1}(edge_{(S)}(x_s, y_s)) \\ &+ \beta \cdot smoothstep_{a0}^{a1}(edge_{(A)}(x_o, y_o)) \end{aligned} \tag{3.10}$$

As shown in Equation 3.10, each of the two edge detection responses is filtered with the $smoothstep$ function. This function is provided by the shading language used for the implementation (see Sec. 3.3.4). It returns a value of zero for edge detection responses below the threshold $s_0$ ($a_0$), and a value of one for responses above $s_1$ ($a_1$). Between the two thresholds, smooth Hermite interpolation is used (see [124] for a complete definition). The parameters $s_0$, $s_1$, $a_0$ and $a_1$ are specified by the user. They determine the minimum edge detection response necessary for generating a silhouette, and how steeply the silhouette intensity increases.

The combined edge detection response $I_o$ is computed for every pixel location $(x_o, y_o)$ in the output image. The final output image is then generated with a simple mixing operation. For every output pixel, the corresponding simplified color image pixel is looked up with an interpolated texture access to $S(x_s, y_s)$. This pixel is then rendered at $(x_o, y_o)$, possibly with a silhouette edge blended over it. The silhouette edge intensity is computed as the factor $(1 - I_o)$, which is used for scaling the values in the RGB color channels of the output pixel. This way, the output pixel is rendered dark if a large combined edge detection response has been computed. The resulting output image is a magnified version of the simplified color image with black silhouette lines rendered over it. This is illustrated in Figure 3.22, which shows the final output image generated for the original AR frame in Figure 3.20a.

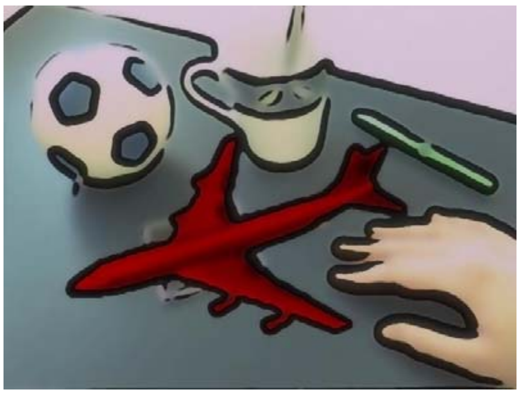


Figure 3.22: The final output of the image stylization filter. (Parameters: $s_0$=0.054, $s_1$=0.064, $a_0$=0.3, $a_1$=0.7, $\beta$=0.3)

### 3.3.4 Implementation Details

The image stylization filtering algorithm for the advanced cartoon-like stylized AR approach was implemented using the OpenGL Shading Language [169]. The shading language makes it possible to execute the code of the stylization filter on the graphics processing unit (GPU). All necessary computations are performed on data which are stored in the local memory of the graphics card. This eliminates the need for a time-consuming readback of graphics memory contents.

The implementation of the algorithm uses three different textures stored in graphics card memory:

- During program startup, a one-dimensional texture storing an exponential function is defined. This exponential texture contains a sequence of function values computed as shown in Equation 3.7. The non-linear filter in the color simplification step looks up the photometric weights for neighboring pixels in this one-dimensional texture. This way, an explicit evaluation of the Gaussian function for every pixel becomes unnecessary. The exponential texture is updated whenever $\sigma_p$ is changed by the user.

- For each frame, the original AR image is copied from the frame buffer into a separate texture. This original AR texture is later used for generating a scaled-down version of the image. Moreover, this texture is later also accessed as image function $\mathbf{A}$ by the edge detection filter for computing the edge detection response $edge_{(A)}$. Standard OpenGL functionality is used for copying the frame buffer content into texture memory (`glCopyTexSubImage2D()`, see [181]).
- The scaled-down version of the original AR image is also stored in a separate texture. This texture is repeatedly overwritten with the results of the iterations of the non-linear filter. After the last iteration, it contains the simplified color image. Since this texture serves as buffer for intermediate images generated by the filtering passes, it is also called *multipass texture* in this thesis. Again, the scaled-down original image and the result images of the filtering passes are copied into the texture using `glCopyTexSub-Image2D()`.

Each of the textures is bound to a separate texture unit of the GPU. This way, they can be accessed simultaneously from the shader programs. The filtering passes are performed by rendering 2D rectangles into the OpenGL back buffer. Before each rectangle is rendered, the respective image filtering shaders are activated. The final result image is also rendered into the back buffer, overwriting data from intermediate passes before the image content of the buffer is displayed to the user.

As mentioned in Sections 3.3.2 and 3.3.3, the image stylization filter converts pixels into the YUV color space. This conversion is performed by multiplying RGB vectors with a constant matrix, which is an operation that can be computed very efficiently on the GPU. An optimized texture access scheme is used for the actual image processing operations. This optimized access method is discussed in the following section.

#### Optimized Texel Addressing Scheme

Both the color simplification and the edge detection stages are implemented as a pair of vertex and fragment shaders. The main processing is performed in the fragment shaders, which require many texture accesses. The color simplification shader accesses a 5 x 5 texel neighborhood. Moreover, the Sobel filter in the edge detection shader reads 3 x 3 texel neighborhoods in the original AR frame and the multipass texture. Each access to a neighboring texel requires the computation of texture coordinates by adding a certain offset to the texture coordinates of the currently regarded image location.

In a straightforward implementation, these new texture coordinates would be computed in a nested loop over the texel neighborhood. In each iteration of the texel loop, the corresponding texture coordinate would be determined by multiplying the loop indices with texture-specific texel size factors. This straightforward implementation requires conditional branches for the loops and one additional vector multiplication per texel. Image processing shaders implemented this way are typically slow, especially since conditional branches are normally not executed efficiently on GPUs. Therefore, the advanced cartoon-like filter uses a method for pre-computing texel addresses which is similar to a technique described in [199].

The basic concept of texel address pre-computation is to utilize the automatic texture coordinate interpolation provided by the GPU. Up to eight texture coordinate vectors, `gl_TexCoord[0]` - `gl_TexCoord[7]`, can be defined simultaneously. Each vector holds four vector components `xyzw`. All of these data are interpolated over the area of a graphical primitive. This is done automatically and does not involve any additional computational costs. The texture coordinate vectors are initialized in the vertex shader with the addresses of neighboring texels. In the fragment shader, the interpolated addresses are then read and used directly for performing texture lookups.

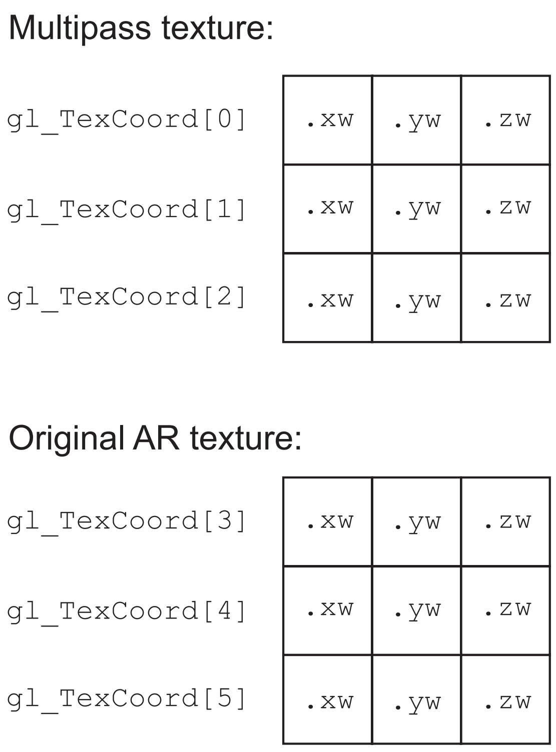


Figure 3.23: Diagram of the scheme used by the edge detection shader for addressing texels in a 3 x 3 neighborhood. Each grid represents a 3 x 3 texel square centered at the currently processed texel.

Figure 3.23 shows an illustration of how the texture coordinate vectors are used in the edge detection shader. Vectors `gl_TexCoord[0]` to `gl_TexCoord[2]` hold the texel addresses for the multipass texture, `gl_TexCoord[3]` to `gl_TexCoord[5]` the addresses for the original AR texture. In both cases, they describe a 3 x 3 neighborhood for the computation of the Sobel filter. As shown in Figure 3.23, different combinations of the components of a texture coordinate vector are required for accessing different texels. In the case of the edge detection shader, the `w` component always holds the texture coordinate `t` for describing the interpolated texel row. The `s` coordinates of texel columns in the same row are stored in the `x`, `y` and `z` components. The arbitrary combination of vector components is directly supported by the OpenGL shading language (*component swizzling*).

As an example, the pre-computation of one line of texel addresses for the multipass texture is shown in Algorithm 4. In this piece of vertex shader code, vector `gl_MultiTexCoord0` holds the input texture coordinates defined for the current vertex. Offsets corresponding to multiples of the distance between texel rows or texel columns are added to this input texture coordinate. The offsets are computed by multiplying integer texel indices with `texelSizeX` or `texelSizeY`. These float values have been previously defined as the texture coordinate distance between neighboring texel columns or texel rows, respectively.

**Algorithm 4** An example for texel address pre-computation (OpenGL Shading Language code fragment.)

```
gl_TexCoord[0].x = gl_MultiTexCoord0.x + (-1 * texelSizeX);
gl_TexCoord[0].y = gl_MultiTexCoord0.x;
gl_TexCoord[0].z = gl_MultiTexCoord0.x + ( 1 * texelSizeX);
gl_TexCoord[0].w = gl_MultiTexCoord0.y + ( 1 * texelSizeY);
```

The same technique is utilized for pre-computing the texel addresses of the 5 x 5 neighborhood required by the color simplification filter. In the fragment shaders, the interpolated texture coordinates are read and their components are combined in order to obtain the addresses of neighboring texels. The texels are then directly accessed using these addresses for computing the weighted summation performed by the image filters.

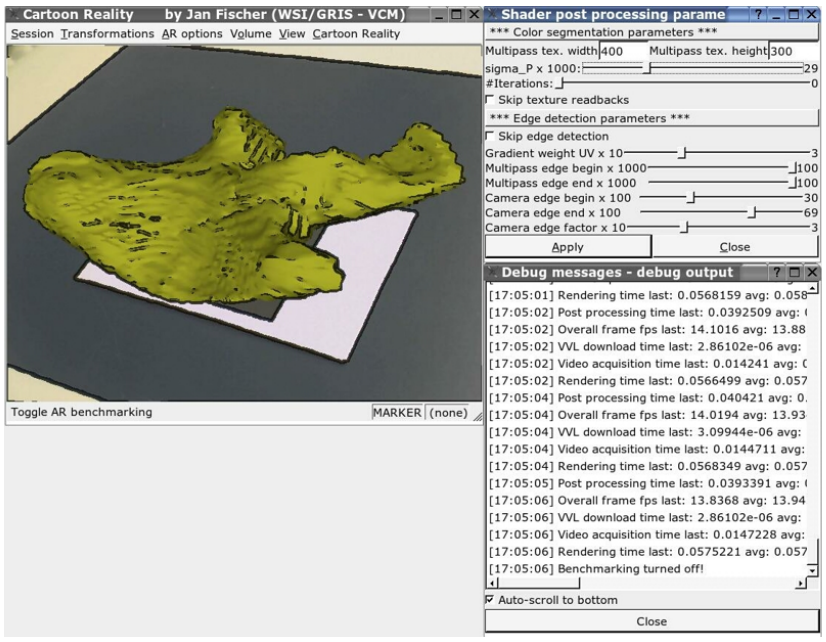
### 3.3.5 Results

The advanced cartoon-like stylization algorithm for augmented video streams was tested with various AR scenes. Like for the original stylization algorithm (see Sec. 3.2.3), an interactive demonstration application was created. The software containing the implementation of the algorithm also provides functionality for importing 3D models in the standard Wavefront OBJ file format. The user can translate, rotate and scale the virtual models in relation to the ARToolKit marker coordinate system. Moreover, material parameters and textures can be assigned to the models. A comprehensive user interface for choosing the parameters of the GPU-based postprocessing filter is also provided. A screenshot of this user interface is shown in Figure 3.24.

Figure 3.25 shows images of three test scenes. In each row of images, the left image contains the original AR frame, and the result of the stylization algorithm is shown in the right image. A virtual Santa Claus model is the virtual object in Fig. 3.25a and 3.25b. In Fig. 3.25c and 3.25d, a virtual Moka Express coffeemaker is located over the ARToolKit marker. The graphical model of a DC10 plane is displayed in the AR scene shown in Fig. 3.25e and 3.25f.

As an additional example application, the visualization of dinosaur bones with the advanced stylized augmented reality method is demonstrated. Figure 3.26 shows three different bone segments rendered in an AR scene using the cartoon-like stylization algorithm. The datasets were generated from CT scans of actual *Plateosaurus* bones.

For rendering the example images in Figure 3.25 and 3.26, the adaptive edge detection was performed with a strong emphasis on the simplified color image. The factor for edges in the original AR frame (parameter $\beta$ in Equation 3.10) was set to values smaller than 0.5. This way, thick silhouette lines between large, homogeneously colored image regions were generated. The selected size of the multipass texture, which is also the size of the final simplified color image, typically was less than half of the dimensions of the original AR frame. Therefore, some detail was removed from the image, and uniformly colored regions were created. As illustrated in Figure 3.25 and 3.26, real and virtual scene elements look very similar in the stylized output video frames.


Cartoon Reality by Jan Fischer (WSI/GRIS - VCM)
Session Transformations AR options Volume View Cartoon Reality
Toggle AR benchmarking
MARKER (none)
Shader post processing parame
Apply
Close
Debug messages - debug output
Close

```
*** Color segmentation parameters ***
Multipass tex. width 400    Multipass tex. height 300
sigma_P x 1000:                29
#Iterations:                    0
Skip texture readbacks
*** Edge detection parameters ***
Skip edge detection
Gradient weight UV x 10         3
Multipass edge begin x 1000   100
Multipass edge end x 1000     100
Camera edge begin x 100        30
Camera edge end x 100          69
Camera edge factor x 10         3
```

```
[17:05:01] Rendering time last: 0.0568159 avg: 0.058
[17:05:02] Post processing time last: 0.0392509 avg: (
[17:05:02] Overall frame fps last: 14.1016 avg: 13.88
[17:05:02] VVL download time last: 2.86102e-06 avg:
[17:05:02] Video acquisition time last: 0.014241 avg: (
[17:05:02] Rendering time last: 0.0566499 avg: 0.057
[17:05:04] Post processing time last: 0.040421 avg: 0.
[17:05:04] Overall frame fps last: 14.0194 avg: 13.93
[17:05:04] VVL download time last: 3.09944e-06 avg:
[17:05:04] Video acquisition time last: 0.0144711 avg:
[17:05:04] Rendering time last: 0.0568349 avg: 0.057
[17:05:05] Post processing time last: 0.0393391 avg: (
[17:05:06] Overall frame fps last: 13.8368 avg: 13.94
[17:05:06] VVL download time last: 2.86102e-06 avg:
[17:05:06] Video acquisition time last: 0.0147228 avg:
[17:05:06] Rendering time last: 0.0575221 avg: 0.057
[17:05:06] Benchmarking turned off!
```

[✓ Auto-scroll to bottom]

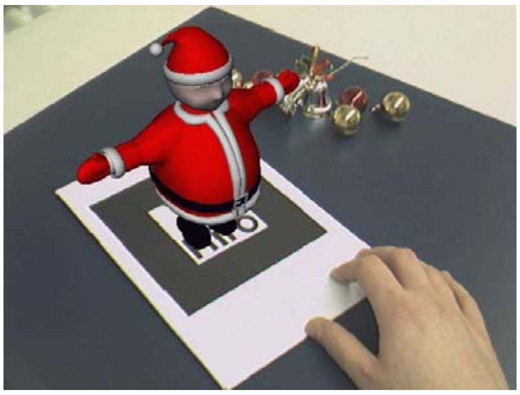
Figure 3.24: The demonstration application for the advanced cartoon-like stylization. The output image of the algorithm as well as the dialog for adapting the parameters of the postprocessing filter are shown.

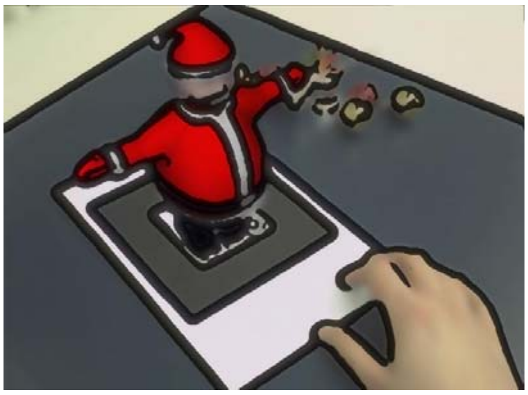
Table 3.3: Runtimes of the advanced cartoon-like stylization filter and overall frame rates for different algorithm parameters.

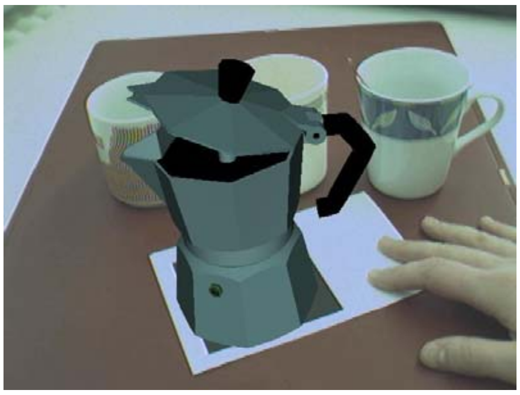
| Resolution (multipass tex.) | Filtering iterations | Postprocessing (msecs) | Overall fps |
|---|---|---|---|
| 240x180 | 5 | 19 | 27.84 |
| 240x180 | 7 | 25 | 24.22 |
| 240x180 | 9 | 30 | 21.51 |
| 400x300 | 5 | 44 | 16.68 |
| 400x300 | 7 | 59 | 13.37 |
| 400x300 | 9 | 74 | 11.12 |

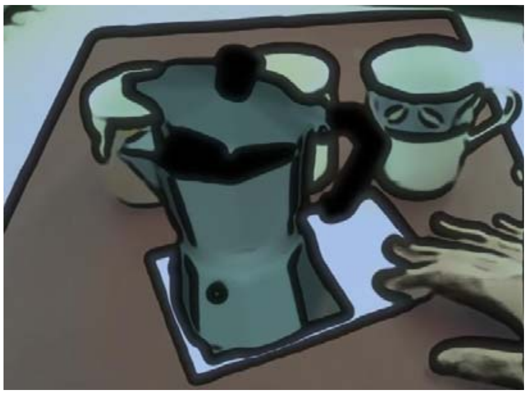
In Table 3.3, frame rates measured with the advanced stylized augmented reality system for different algorithm parameters are listed. These measurements were taken on a computer system with an Intel Pentium 4 Xeon processor running at 2.66 GHz and a graphics card based on an NVidia GeForce FX 6800 GT chipset. The webcam used in the AR system delivers video images with a resolution of 640 x 480 pixels[1]. The first column in the table lists the selected resolution of the multipass texture (*shrunkImageWidth* x *shrunkImageHeight*). In the second column, the number of non-linear filter iterations (*numFilterSteps*) is listed. The third column contains the measured runtimes of the image stylization filter in milliseconds. Finally, column four shows the overall system frame rate including the entire augmented reality pipeline.

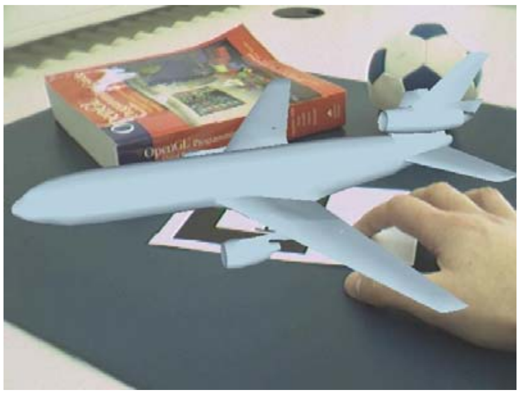
[1]The results shown in this thesis were obtained with Firewire (IEEE 1394) webcams delivering a resolution of 640 x 480 pixels at a rate of 30 Hz. For the development and the experiments, either *Unibrain Fire-i* or *Pyro Firewire* webcams were used.


(a) Santa Claus model, conventional AR


(b) Santa Claus model, cartoon-like style


(c) Moka Express model, conventional AR


(d) Moka Express model, cartoon-like style


(e) DC10 plane model, conventional AR

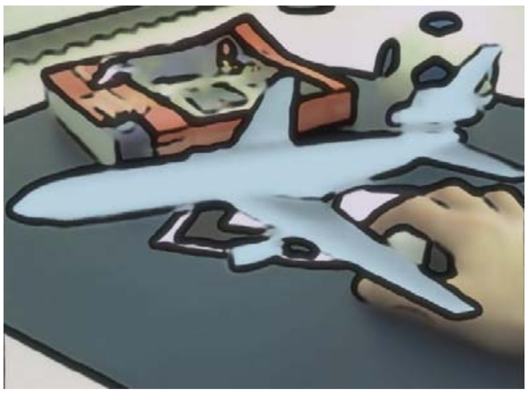
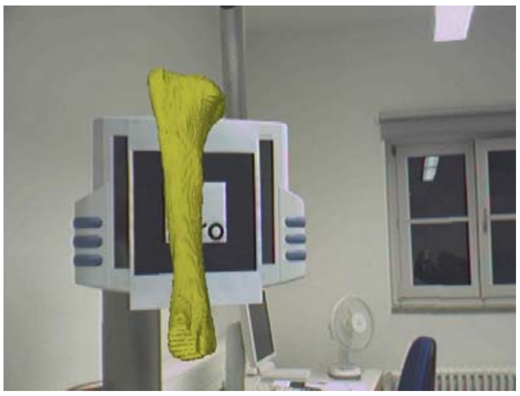
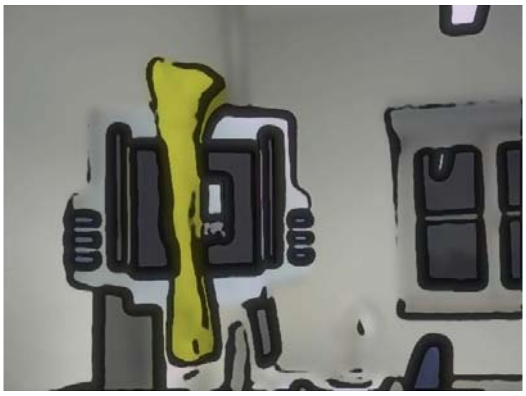
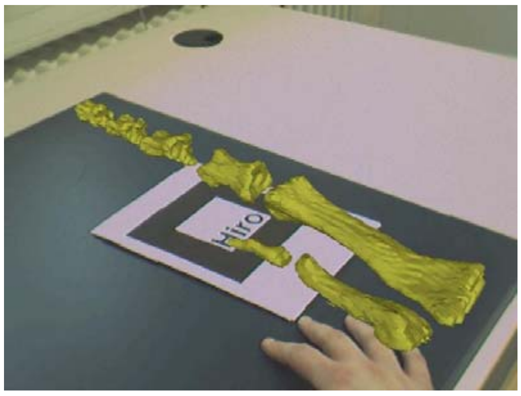

(f) DC10 plane model, cartoon-like style

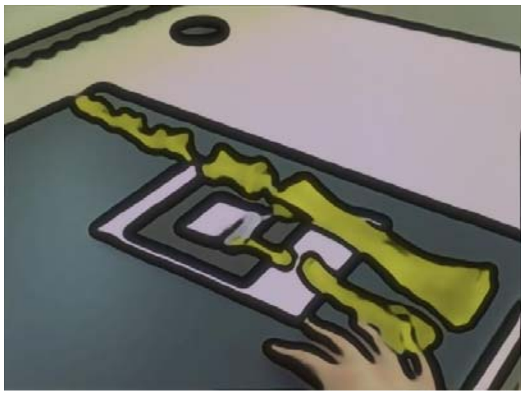
Figure 3.25: Three example scenes illustrating the effect of the GPU-based cartoon-like stylization for augmented reality. In each row, the left image shows conventional AR rendering. The right column contains the stylized versions of the respective video frames.

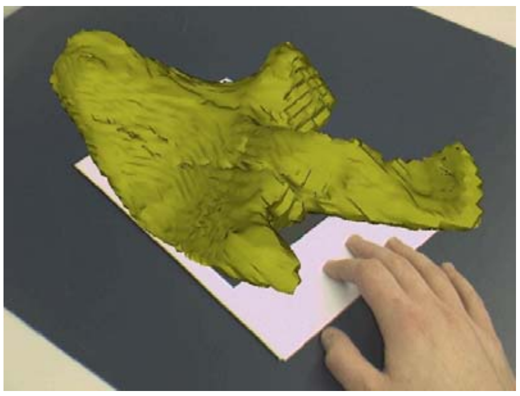

(a) *Plateosaurus* bone 1, conventional AR


(b) *Plateosaurus* bone 1, cartoon-like style



(c) *Plateosaurus* bone 2, conventional AR


(d) *Plateosaurus* bone 2, cartoon-like style


(e) *Plateosaurus* bone 3, conventional AR

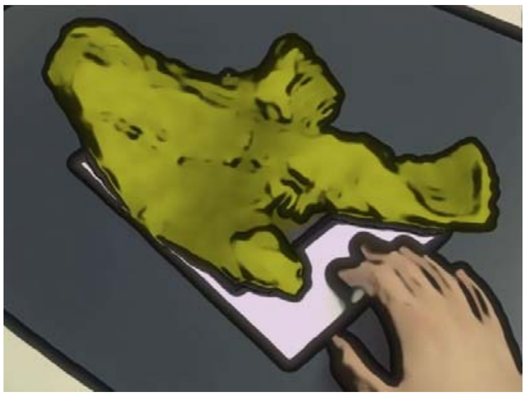
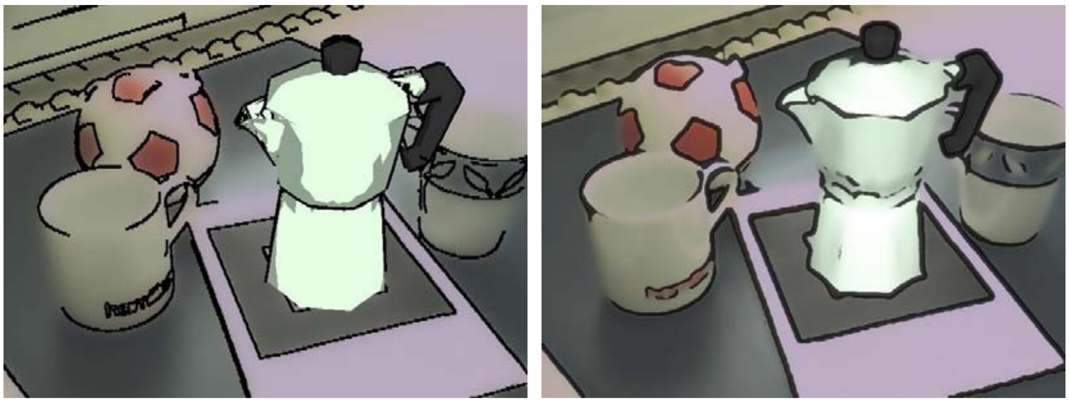

(f) *Plateosaurus* bone 3, cartoon-like style

Figure 3.26: Visualization of dinosaur bones in stylized augmented reality. Using the advanced cartoon-like stylization, the virtual bone models look very similar to the real environment.

A multipass texture resolution of 240 x 180 texels was empirically found to be sufficient for generating an output video stream of good visual quality in most cases. Moreover, no more than 7 filter iterations are usually necessary. Assuming that the graphical models contained in the AR scene do not consist of an excessive number of polygons, the GPU-based cartoon-like system can generate stylized augmented video streams at 25 fps or more in a typical setup.

### Comparison of the two Cartoon-like Stylization Algorithms

The GPU-based stylization method for AR described in this section generates output images of a high visual quality. This is illustrated in Figure 3.27, which compares output images rendered by the advanced method and the original cartoon-like stylization algorithm described in Section 3.2. The image generated by the previous algorithm shown in Figure 3.27a contains aliasing, and some important silhouette lines are missing. In the video frame rendered by the GPU-based method (Fig. 3.27b), the colors in the camera image are better preserved, and a greater similarity between the real background and the virtual object is created.



(a) Moka Express scene, original algorithm (b) Moka Express scene, GPU-based algorithm

Figure 3.27: Comparison of the output generated by the original algorithm and the GPU-based method for an example scene.

The original cartoon-like algorithm is typically only capable of delivering frame rates of approximately 15 fps (see Sec. 3.2.3). The advanced GPU-based method can normally render a stylized augmented video stream with more than 25 frames per second (see Table 3.3).

## 3.4 Brush Stroke Stylization of AR Images

In addition to the cartoon-like stylized AR systems described above, a second type of stylization for augmented reality was developed in the context of this thesis [11, 37]. This alternative approach mimics an artistic style found in paintings. The generated output video frames consist of a large number of small brush strokes. This rendering method recreates the technique applied by painters who adhere to the *pointillism*[2] style of painting [42].

The design of the brush stroke stylization system aims at fulfilling the requirements listed in Section 3.1. The algorithm tries to create output images in which the virtual elements and the background camera image are represented in a visually similar style. Moreover, the system achieves interactive frame rates when generating the output video stream. Like the original cartoon-like stylized AR approach (see Sec. 3.2), the brush stroke method uses two different processing branches in the image generation pipeline; one branch for the camera image, a second branch for the virtual objects.

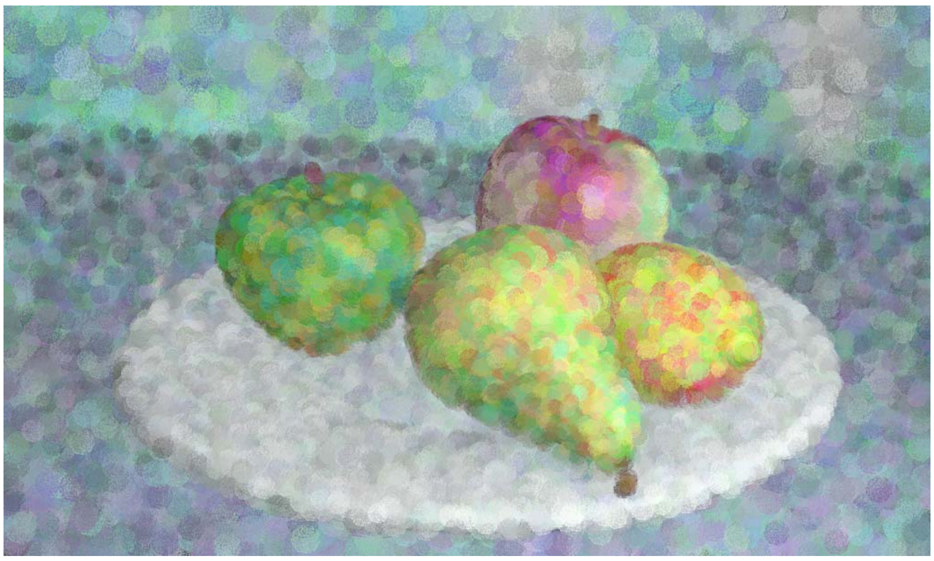


Figure 3.28: An example image generated with Meier's original algorithm (from [145]).

The rendering algorithm used by the brush stroke stylized AR system is inspired by a non-photorealistic method described by Meier [145]. Meier presented an approach for the painterly rendering and animation of completely virtual scenes. This method is based on two main ideas. As the first central step, the graphical models are converted into a particle representation. During the actual rendering process, these particles are then projected into screen space and interpreted as the positions of brush strokes. As the second main step, a so-called reference image is rendered from the original scene description for each frame as the basis for the visual properties of the individual brush strokes (e.g., stroke color). An example image rendered with Meier's algorithm is shown in Figure 3.28. Many derivations and improvements of this original brush stroke rendering method exist, including Sperl's system, which is a speed-optimized variation of the algorithm [182].

[2]It has to be noted, however, that the use of small brush strokes is only one aspect of *pointillism* in the strict sense. Another important feature the *pointillism* style of painting is the fact that only primary colors are used for the tiny brush strokes, generating secondary colors as an optical effect. In this sense, the technique presented here emulates only one aspect of the *pointillism* style of painting.

The main challenge in applying the brush stroke rendering algorithm to stylized augmented reality is the filtering of the camera image. Since the original approach described by Meier can only process graphical models completely defined in 3D, it cannot be used for stylizing two-dimensional image data. Therefore, an adapted method for generating a brush stroke representation of camera images was developed in the context of this thesis. Moreover, the rendering method for virtual objects had to be adapted significantly in order to achieve a visually compatible output for real and virtual scene elements.

In the remainder of this section, an overview of the brush stroke system is given in Section 3.4.1. Sections 3.4.2 and 3.4.3 discuss the camera image filter and the renderer for virtual objects, respectively. Finally, results generated with the brush stroke stylization system are presented in Section 3.4.4.

### 3.4.1 Overview of the Brush Stroke System

In brush stroke stylized AR, a painterly filter is applied to the input camera image. This painterly filter randomly samples the camera image and paints brush strokes with the colors of the sampled camera pixels. Afterwards, the virtual model is rendered in the brush stroke style. In order to create an appearance similar to the processed camera image, the representation of the virtual model also consists of colored brush strokes. The polygonal geometry of the virtual model is first converted to a three-dimensional particle model, which is then used for the actual painterly rendering. The renderer projects each of the particles into screen space and uses these 2D positions as basis for the brush stroke representation of the model. Therefore, the painterly rendering method can be considered a simplified specialization of point based rendering (e.g., see [161]).

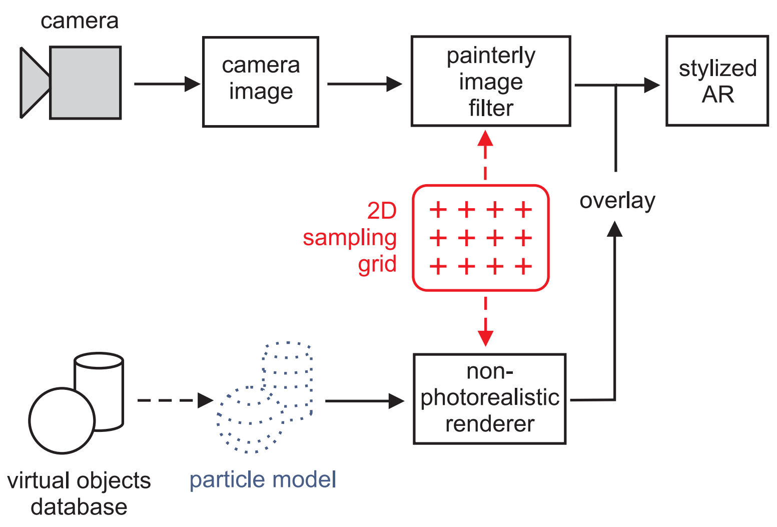


Figure 3.29: Overview of the brush stroke stylized AR image generation pipeline.

It is an important observation that the same brush stroke positions must be used in order to make the virtual model and camera image look similar. In early experiments employing a method which directly renders projected particles, the brush stroke rendering of the virtual model appeared to float above the background image. The newly developed method therefore assigns each projected particle position to the nearest point in a precomputed 2D brush stroke grid. This is the same grid that is used for sampling the camera image. Figure 3.29 shows an overview of the system for brush stroke stylization.

### 3.4.2 Brush Stroke Filter for Camera Image

#### Generation of 2D Sampling Grid

During the initialization of the brush stroke stylized AR software, a two-dimensional sampling grid is generated in a one-time preprocessing step. The grid remains fixed throughout the runtime of the system. It is stored as an array of sampling point records. Each sampling point record contains the 2D position of the point and additional information about the brush stroke which is to be painted there.

Table 3.4: Attributes stored for one sampling point.

| **Attribute** | **Data type** |
|---|---|
| $x_{sample(i,j)}$ | integer |
| $y_{sample(i,j)}$ | integer |
| $radius_{(i,j)}$ | integer |
| **colorOffset**$_{(i,j)}$ | RGB color vector |

Table 3.4 lists the attributes stored for each sampling point. $x_{sample(i,j)}$ and $y_{sample(i,j)}$ are the two-dimensional position of the point in the camera image, and $radius_{(i,j)}$ is the radius of the brush stroke to be drawn there. Moreover, a fixed RGB color offset is stored for each sampling point. This color offset is later added to the color of any brush stroke rendered at that position.

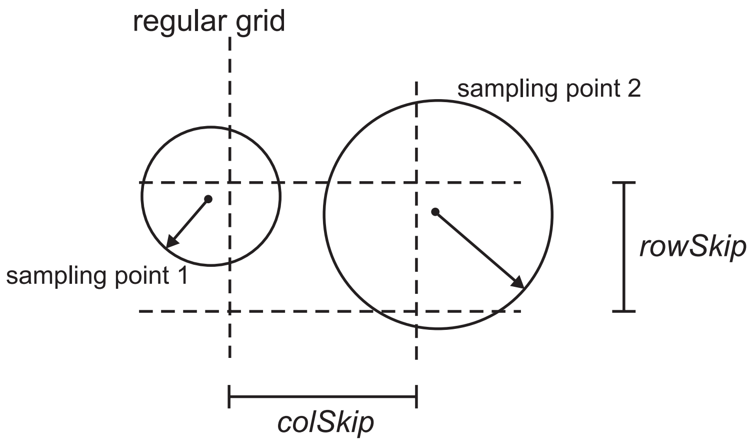


Figure 3.30: Illustration of two sampling points randomly displaced from the regular grid with random brush stroke radii.

Sampling points are generated over the entire area of the camera image. Each point is initially located on a regular grid with a horizontal step size of $colSkip$ and a vertical step size of $rowSkip$. A random displacement vector is added to the point positions, as shown in Equation 3.11. The maximum random offset $range$ is a user-definable parameter. Brush stroke radius $radius_{(i,j)}$ and RGB color offset **colorOffset**$_{(i,j)}$ are also generated randomly with random number ranges that can be modified by the user. An illustration of two sampling points is shown in Figure 3.30.

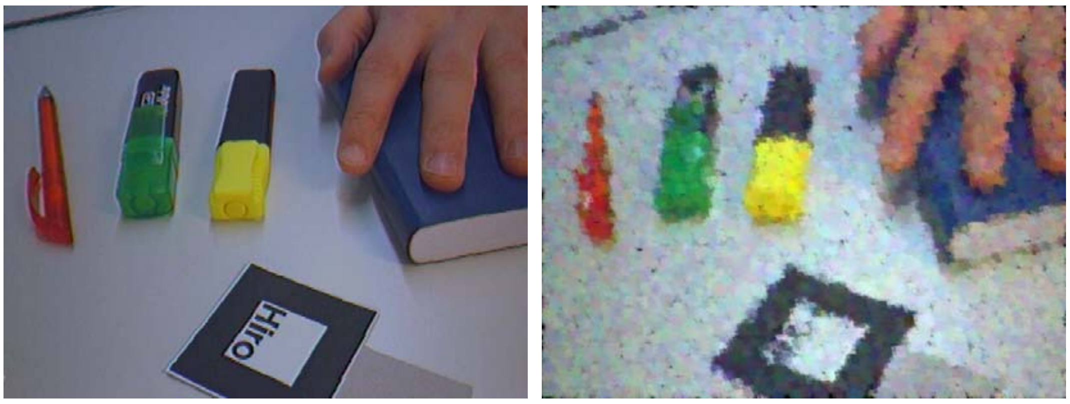
$$\begin{aligned} x_{sample(i,j)} &= i \cdot colSkip + rand(-range, range) \\ y_{sample(i,j)} &= j \cdot rowSkip + rand(-range, range) \end{aligned} \tag{3.11}$$

The random variations of point position, brush stroke radius and brush stroke color are introduced into the sampling point grid in order to create a more natural, irregular look in the generated image. Note that the total number of grid points depends on $colSkip$ and $rowSkip$. The random number range for the $radius_{(i,j)}$ has to be selected so that a good coverage of the image area with brush strokes is achieved.

Another measure for creating a more natural look for the processed image is the application of a random drawing order of the brush strokes. An array containing the indices of all sampling points is generated. This array is then randomly shuffled. When the camera image filter is applied, this index array is traversed sequentially, resulting in a random brush stroke drawing order.

### Camera Image Filter

The camera image filter samples the input image by reading pixel colors at the sampling point positions in the given, random order. The color offset **colorOffset**$_{(i,j)}$ is then added to each pixel color, and the resulting RGB components are clamped to the valid real number range $[0; 1]$. Each brush stroke is drawn as a textured square with side length $2 \cdot radius_{(i,j)} + 1$, centered at $(x_{sample(i,j)}, y_{sample(i,j)})$. The brush stroke texture is loaded from file during the initialization of the system. Alpha blending is enabled to achieve partial transparency when rendering overlapping brush strokes. An example of a brush stroke image generated by the camera image filter is shown in Figure 3.31.



(a) Original camera image (b) Result of brush stroke filter

Figure 3.31: Example of an image generated by the brush stroke filter for the camera image.

### 3.4.3 Brush Stroke Renderer for Virtual Objects

#### Creation of Particle Model

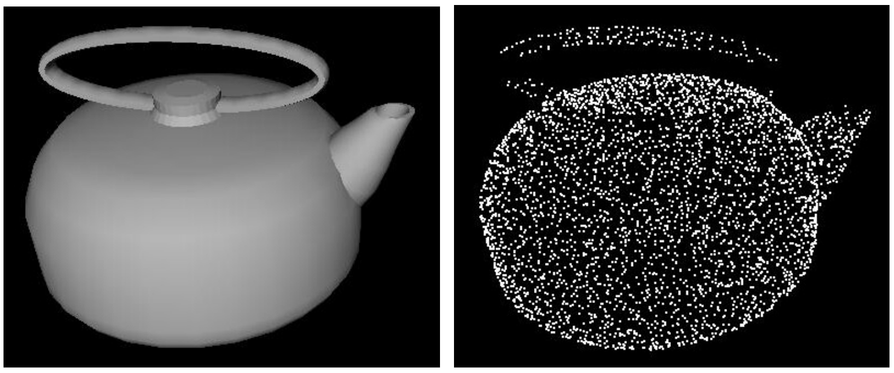
Virtual objects in the augmented scene are first converted into particle models before they can be rendered in the brush stroke style. For each of the polyons constituting a graphical object, a number of particles lying on the polygon is generated. Each three-dimensional particle position is computed as a weighted sum of the polygon vertices. In Equation 3.12, the $\mathbf{v}_i$ denote the vertices of the current polygon, and $N$ is the number of vertices. The position of the currently regarded particle is called $\mathbf{particlePos}_j$.

$$\mathbf{particlePos}_j = \sum_{i=1}^{N} w_i \cdot \mathbf{v}_i \qquad \sum_{i=1}^{N} w_i = 1 \tag{3.12}$$

As expressed in Equation 3.12, the vertex weights $w_i$ sum up to a value of one, so that $\mathbf{particlePos}_j$ is located on the polygon. The weights are chosen randomly in order to generate a random particle position. Particle color $\mathbf{particleCol}_j$ and normal vector $\mathbf{particleNorm}_j$ are stored as additional attributes for each particle. The particle color is obtained from a texture lookup, for which texture coordinates are computed as a sum of the texture coordinates of the polygon vertices weighted with $w_i$. The user can load any bitmap as texture image for the graphical model. The particle normal is calculated by correspondingly interpolating the normals of the polygon vertices and normalizing the result.

In order to achieve a homogeneous distribution of particles over the entire surface of the model, a specific number of particles is calculated for each polygon. The total number of particles to be generated is selected by the user as algorithm parameter $numParticles$. Before generating the particle model, the total surface area of the virtual model, $totalArea$, is determined. For each polygon, the number of particles is then computed as the ratio of its area to $totalArea$ multiplied by $numParticles$.

An example of a particle model generated for a virtual object is shown in Figure 3.32.



(a) Polygonal model (b) Particle model

Figure 3.32: Particle model generated for a teapot object.

### Sampling Point Lookup Table

During the brush stroke rendering process, the nearest 2D sampling point has to be determined for each projected particle. The computational complexity of this task is large if a naive search strategy is used. Since a high overall rendering speed is required, a lookup table containing the index of the nearest sampling point for each pixel in the camera image is computed. This table can be considered a representation of the Voronoi diagram of the sampling point set [208]. An illustration of an example lookup map is shown in Figure 3.33. The lookup table is generated in a one-time preprocessing step during the initialization of the brush stroke stylized AR software.

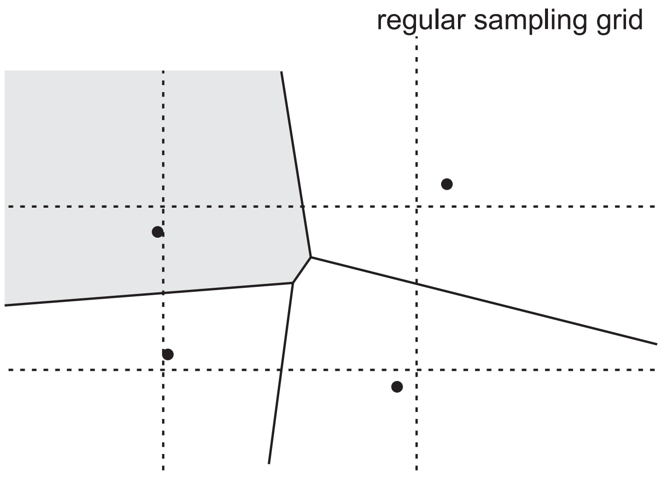


Figure 3.33: Illustration of the Voronoi diagram of four sampling points. The black dots are the sampling point positions, randomly displaced from the regular grid. The gray area in the top left part represents the camera image pixels which are assigned to the top left sampling point. The three remaining segments belong to the respective other sampling points.

The main challenge in computing the lookup map is the fact that the random displacement of sampling points can be large. Therefore, the nearest sampling point cannot easily be determined for a given camera image pixel. The straightforward approach, namely traversing the camera image and computing the distance to each sampling point from the current pixel coordinates, is extremely inefficient. This simple method has proven to be too slow even for a preprocessing step. Even for a relatively small number of points, this method can require a runtime of several minutes. Hence, the reverse approach to generating the lookup map was implemented. For each sampling point, the distances to camera pixels in a neighbourhood are compared with their distances to adjacent sampling points.

$$\begin{gathered} dist_{ij}(x,y) = \sqrt{(x_{sample(i,j)} - x)^2 + (y_{sample(i,j)} - y)^2} \\ \text{for currently regarded } (i,j)\text{:} \\ \mathbf{map}(x,y) = \quad arg \quad \underset{\substack{i'=i-1,\dots,i+1 \\ j'=j-1,\dots,j+1}}{min} \quad dist_{i'j'}(x,y) \end{gathered} \tag{3.13}$$

The points in the sampling grid are processed consecutively. Assuming that the grid indices of the current sampling point are $(i,j)$, the lookup map algorithm evaluates Equation 3.13

for each pixel position $(x, y)$ in a pixel neighbourhood. This means that lookup table entry **map**$(x, y)$ will contain the grid position of the nearest of the 3 x 3 adjacent sampling points.

The size of the regarded pixel neighbourhood depends on the grid step sizes $rowSkip$ and $colSkip$ as well as the maximum random displacement $range$, as shown in Equation 3.14. The neighbourhood of pixel $(x, y)$ is defined as a rectangle which extends horizontally from $(x - xExtent)$ to $(x + xExtent)$ and vertically from $(y - yExtent)$ to $(y + yExtent)$.

$$\begin{aligned} xExtent &= (\tfrac{colSkip}{2} + range) \\ yExtent &= (\tfrac{rowSkip}{2} + range) \end{aligned} \tag{3.14}$$

The described reverse approach to computing the sampling point lookup map results in a significantly faster intialization stage of the system. Table 3.5 lists computation times of the lookup map for a 640 by 480 pixels camera image with typical sampling grid parameters. These data were measured on a PC with a Pentium 4 processor running at 2.8 GHz.

Table 3.5: Typical computation times for the sampling point lookup map.

| $rowSkip$ | $colSkip$ | $range$ | **Runtime (secs.)** |
|---|---|---|---|
| 3 | 3 | 2 | 2.47 |
| 3 | 3 | 3 | 4.06 |
| 5 | 5 | 3 | 2.16 |
| 5 | 5 | 5 | 3.97 |
| 7 | 7 | 6 | 3.25 |

**Rendering Process**

The actual rendering procedure for the particle models first determines the current color for each particle, projects the particles into screen space and sorts the particles according to their depth. The sorted particle list is then rendered from back to front, taking into account the positions and parameters of the 2D sampling points.

The renderer rotates the particle normals **particleNorm**$_j$ according to the current transformation matrix. The active transformation and projection matrices as well as viewport parameters are retrieved from the current OpenGL state at the beginning of the rendering process. Based on the transformed normal vectors, particles on back-facing polygons are culled from the list of particles to be rendered. For visible particles, the current brightness is computed with diffuse reflection using the normal vector information (see Eq. 3.5). These brightness values are used for scaling the particle colors **particleCol**$_j$ in order to achieve a shaded look for the model.

In the next step of the rendering process, the front-facing particles are projected into screen space on the basis of the stored matrices and viewport parameters. The resulting $x$ and $y$ coordinates and depth values are stored for each particle. Subsequently, the particle list is sorted according to the depth values.

Finally, brush strokes are rendered for the projected particles in the order of descending depth values. This drawing sequence constitutes an implementation of the *painter's algorithm* for the correct mutual occlusion of brush strokes [90]. It is necessary because Z-buffer tests have to be disabled for rendering the semi-transparent brush strokes with alpha blending.

The nearest 2D sampling point for each projected particle at $(x, y)$ is looked up in the precomputed table entry $\mathbf{map}(x, y)$. The brush stroke is then drawn with the same technique as used by the camera image filter (see Section 3.4.2): The color offset stored in the sampling point record is added to the particle color, and then a semi-transparent textured rectangle with a side length depending on $radius_{(i,j)}$ is rendered at the sampling point location. Therefore, a brush stroke drawn for a model particle has the same visual properties as a brush stroke drawn by the camera image filter.

Figure 3.34 shows an example of a virtual 3D model drawn by the brush stroke renderer.

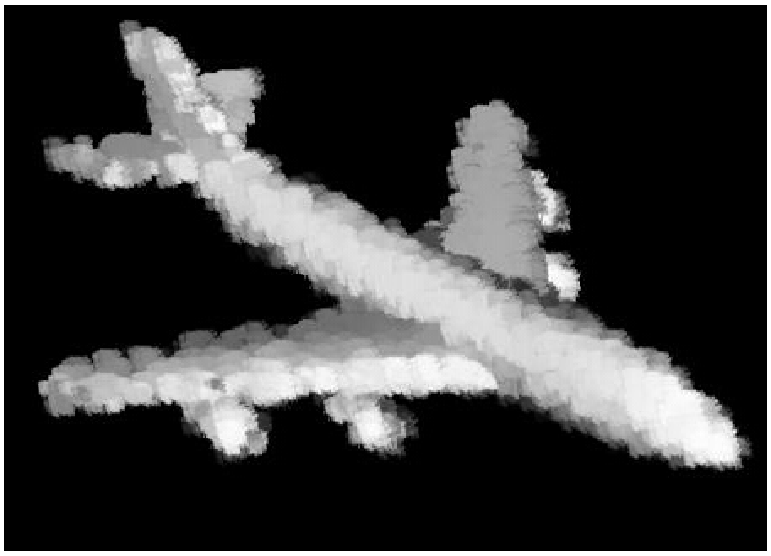
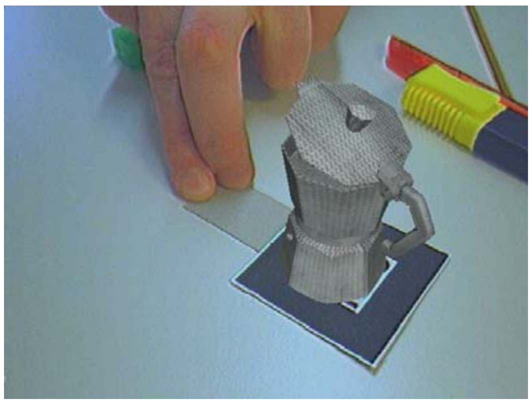


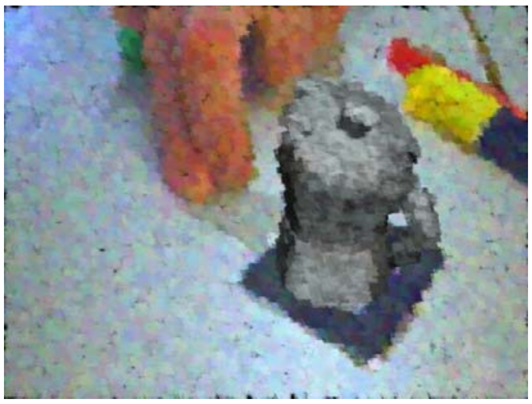
Figure 3.34: Brush stroke rendering of a uniformly colored plane model with diffuse lighting.

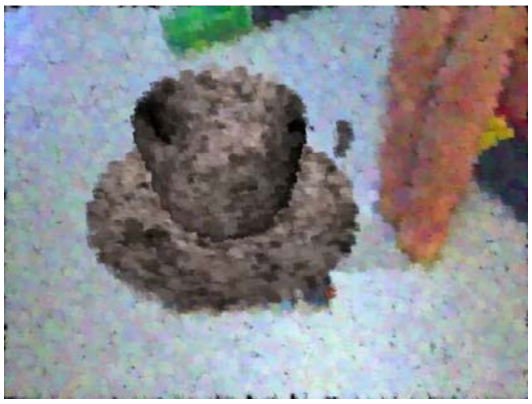
### 3.4.4 Results

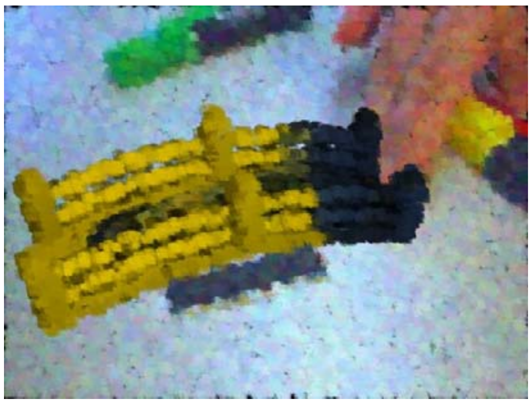
The brush stroke stylization method for augmented reality was tested with various example scenes. Like the stylization systems described above, the brush stroke stylized AR software contains an editor which is capable of importing 3D models in the standard Wavefront OBJ file format. Again, the user can freely place, scale and rotate the model. Bitmap files can be loaded and applied as textures to the virtual models. The augmented reality display can be switched interactively between conventional AR and the brush stroke mode at any time. For all the test scenes shown here, optical marker tracking based on the ARToolKit framework was used [123].

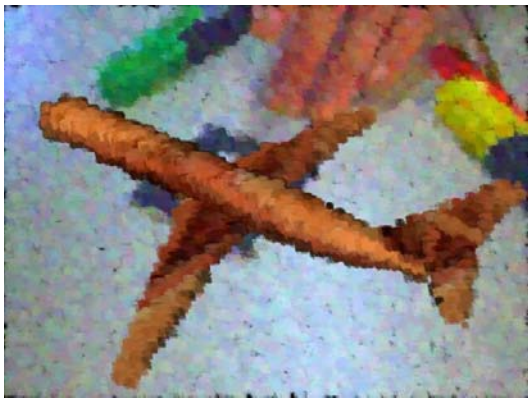
Images rendered with the brush stroke stylization algorithm are shown in Figure 3.35. Figures 3.35a and 3.35b compare the conventional and brush stroke representation of an AR scene containing a Moka Express coffee maker as the virtual object. In Figure 3.35c, a stylized augmented reality scene containing a virtual teacup with a marble texture is depicted. The scene shown in Figure 3.35d comprises a bridge model with a painting applied as texture. A wooden plane model is the virtual object in Figure 3.35e. Finally, Figure 3.35f shows a virtual beer bottle with an artificial marble texture applied to it.


(a) Moka Express (metal), conventional AR


(b) Moka Express (metal), brush stroke style


(c) Teacup (marble), brush stroke style


(d) Bridge (painting), brush stroke style


(e) Plane (wood), brush stroke style

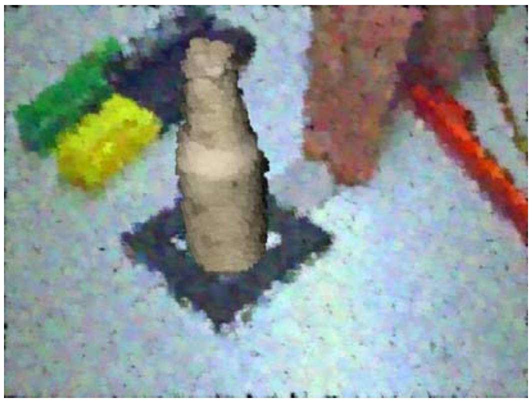

(f) Beer bottle (marble), brush stroke style

Figure 3.35: Five example scenes demonstrating the brush stroke stylization method. The description of the scenes contains the type of texture applied to the virtual model in brackets.

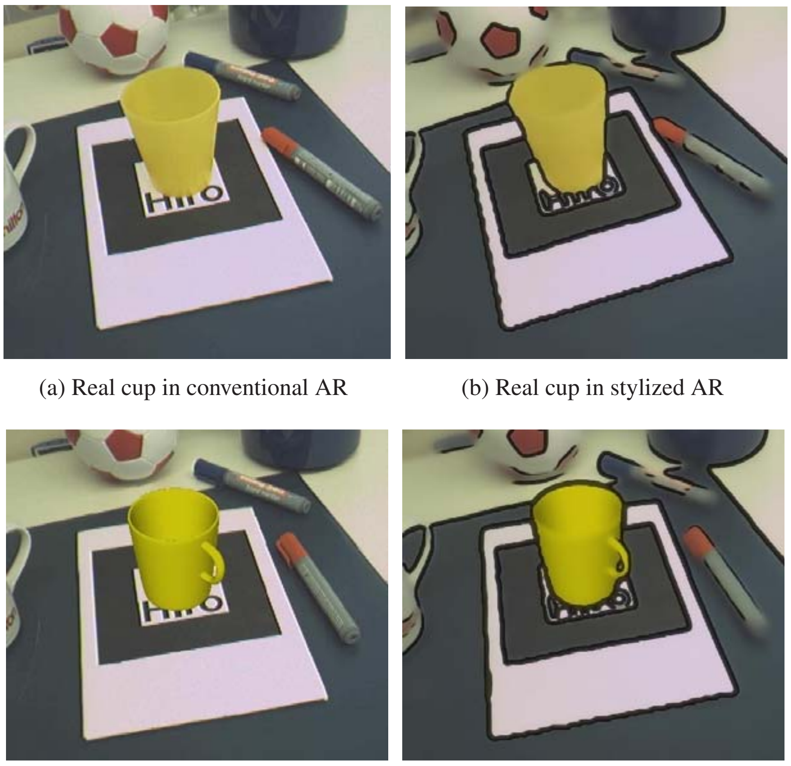
Benchmark measurements show that the brush stroke stylization system delivers an average overall frame rate of more than 14.5 fps with typical scene generation parameters. This measurement was taken on a computer with an Intel Pentium 4 Xeon processor running at 2.66 GHz and a graphics card based on an NVidia GeForce FX 6800 GT chipset. The augmented reality system uses a Firewire webcam delivering a resolution of 640 by 480 pixels. The measurements show that the brush stroke stylization algorithm is capable of delivering a video stream at interactive frame rates.

## 3.5 Study on the Discernability of Virtual Objects in Stylized AR

In the preceding sections, several realizations of the principle of stylized augmented reality were described. The basic concept of stylized AR is to generate an output video stream in which real and virtual scene elements are difficult to distinguish. The decreased - or even non-existent - discernability of virtual objects leads to a novel experience for the users of such a system. A well-designed stylized AR application might even result in a stronger feeling of presence in the augmented environment. If physical and graphical objects cannot easily be distinguished anymore, users will probably tend to impute the same significance to virtual elements in an augmented environment as to real objects. The feeling of presence in virtual and augmented reality, however, is difficult to measure. A considerable amount of previous work exists on attempts to quantify presence in VR and AR (e.g., see [167, 206]).

In order to examine the effectiveness of the stylized AR approach, a basic psychophysical study was performed in the context of this thesis [16]. In this study, it is assumed that the elementary evidence for the effectiveness of stylized AR is the degree of difficulty of distinguishing real objects from virtual models. This means that in stylized augmented reality, it should be measurably more difficult for the user to tell whether an object visible in the augmented image is virtual or not. This concept is illustrated in Figure 3.36: Figure 3.36a shows a real cup in a conventionally rendered image. In this image - and even more so in an interactive real-time AR setup - it is relatively easy to identify the cup as an actual physical object. Figure 3.36b, by contrast, contains a image of the same real cup stylized with the GPU-based cartoon-like filter described in Section 3.3. In Figure 3.36d, the stylized augmented reality rendering of a similarly shaped virtual cup, which has been geometrically modelled, is shown. In the latter two, it should be more difficult for an observer to tell if the central object is real or virtual.

For the sake of clarity, the following terminology is used in this section with regard to the psychophysical study. Actually existing items in the environment of the user which are visible in the camera image are called “physical objects” (e.g., see Fig. 3.36a and 3.36b). The term “virtual objects” is used for computer generated graphical models in the augmented image (see Fig. 3.36c and 3.36d). This distinction between physical and virtual objects is denoted as *object type*. As a concept which is orthogonal to these two types of object, two types of augmented reality rendering are also distinguished. Both physical and virtual objects can be displayed in “stylized” (see Fig. 3.36b and 3.36d) and “conventionally rendered” (see Fig. 3.36a and 3.36c) augmented reality images. The difference between conventionally rendered and stylized is called *AR rendering style*.



(a) Real cup in conventional AR (b) Real cup in stylized AR

(c) Virtual cup model in conventional AR (d) Virtual cup model in stylized AR

Figure 3.36: Two example objects used in the study: A real cup (Fig. 3.36a and 3.36b) and a virtual cup model (Fig.3.36c and 3.36d). All test objects in the study are located directly over the marker used for camera tracking. In order to provide some visual reference for the participants, several background objects are placed near the marker. For each test object, conventionally rendered as well as stylized AR images were recorded and presented to the participants.

The remainder of this section is structured as follows. Section 3.5.1 gives an overview of related work on psychophysics for virtual and augmented environments. The experimental methodology of the study is discussed in Section 3.5.2. Section 3.5.3 presents the experimental results of the psychophysical study. Finally, Section 3.5.4 summarizes the conclusions which can be drawn from the study.

### 3.5.1 Related Work

Several researchers have performed user-based studies for evaluating different aspects of virtual and augmented reality systems. Pausch et al. presented a work on quantifying immersion in virtual reality [159]. The connection between visual information and haptic feedback in a VR system was examined by Burns et al. [66]. Gabbard et al. have presented a study on the suitability of text drawing styles in an augmented reality application [94].

The experiment described in this section is a psychophysical examination of the effect of stylization on the perception of realism. A number of different techniques have been used to determine the perceptual realism of computer graphics algorithms [72, 136, 140, 141, 144, 165, 201]. Some experiments rely on indirect measures, examining the "behavioral realism".

In these experiments, performance on a specific task is compared between the real world and a virtual environment. The similarity of the responses produced by the virtual and real scenes is a measure of the behavioral realism of the system. For example, Mania and Robinson [141] compared estimates of presence and subjective lighting quality for both real world scenes and several virtual environments. Presence was measured using standard questionnaires and the subjective response to lighting was measured using semantic differentials (e.g., the scene was rated along several dimensions, such as warm versus cold and relaxing versus tense, using a scale of 1 to 79).

Other experiments determine the realism of a scene more directly [136, 165]. Longhurst et al. [136], for example, showed a series of carefully controlled real scenes, photographs of the real scenes, and printed photographs of rendered versions of the real scenes to participants and asked them a variety of questions, including "was the image real?". By systematically varying scene properties, such as shadows and lighting quality, these experiments can determine the impact of those properties on perceived realism.

### 3.5.2 Experimental Methodology

The psychophysical study described in this section was designed as an offline task. This means that the participants did not wear a head-mounted display driven by an interactive augmented reality application. Optical marker tracking, which is used by the AR framework developed in the context of this thesis, often fails in an interactive setting with inexperienced users because they tend to accidently occlude the marker or move it out of the camera image. This leads to the inadvertent disappearance of all virtual objects, revealing that they cannot be physical. Moreover, such a setup would have complicated the execution of the study. Since an assessment of visual differences was the main objective of the study, only recorded still images and short video clips were shown to the participants on a conventional monitor.

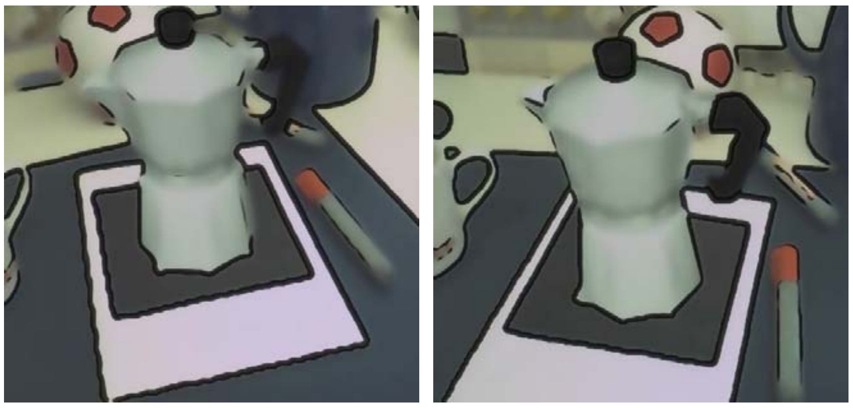

(a) First frame of video clip (b) Last frame of video clip

Figure 3.37: First and last frame of an AR video clip showing the stylized rendering of a virtual coffeemaker.

The test scenes constituting the psychophysical study show different virtual and physical objects. For each object, one conventionally rendered and one stylized video clip were recorded by grabbing real-time frame buffer images from the actual augmented reality application. The GPU-based cartoon-like stylization method described in Section 3.3 was used for producing the stylized images and video clips shown in the study. A standardized setting consisting of a large optical marker with some real background objects was used when recording the video clips (see Fig. 3.36). The currently regarded object is always centered directly over the marker. Each video was shot with a standardized camera path. The beginning and end of this camera movement are illustrated in Figure 3.37. The first frame of each video clip was also used as still image for the static experiments. The recorded video clips and still images were presented to the participants. They were asked to decide whether the displayed object is physical or virtual. The correctness of the response and the participant's reaction time were recorded in a protocol file and later evaluated.

### Outline of the Study

Conventional and stylized presentations of 15 physical and 15 virtual objects were presented to 18 individuals in a psychophysical experiment. The individuals, who participated in return for financial compensation at standard rates, were randomly assigned to one of two groups. One group was presented with a video sequence of the camera moving around the object (the "Dynamic" group). The other group was presented with the first frame of the video sequence (the "Static" group). The participants' task was to determine if the central object in each image was physical or virtual. A selection of images of objects used in the study is shown in Figure 3.38.

### Stimuli

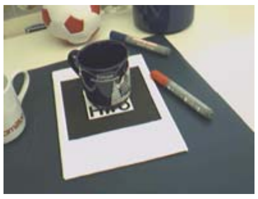
Using the AR framework and the real-time stylization algorithm described in Section 3.3, the 15 real and the 15 virtual objects were recorded using a simple, partially curved camera trajectory (see Figure 3.37). Each object was filmed both in conventional mode as well as in stylized mode, yielding 60 recordings. Care was taken so that the trajectory of the camera was as identical as possible across the 60 recording sessions. The video sequences were approximately 4 seconds each. During the study, the resulting images were scaled from their original size of 640 x 480 pixels to 1024 x 768 pixels, filling the screen of the 21 inch computer monitor. Since the participants sat at a distance of approximately 0.5 meters from the computer screen, the images subtended approximately 43.6 by 33.4 degrees of visual angle[3].

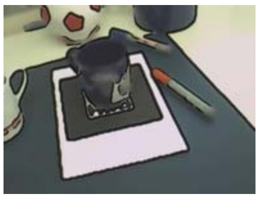
### Procedure

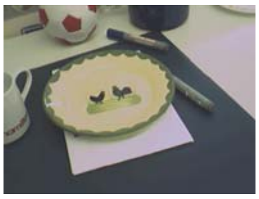
The participants were given an instruction sheet describing the experimental task. In particular, the participants were told that they would be presented with several images containing a tracking marker, on top of which would be either a physical object or a virtual object. They were given several example photographs[4]. The participants were also told that sometimes the images would be rendered in conventional manner and sometimes in a stylized manner, and were again given examples.

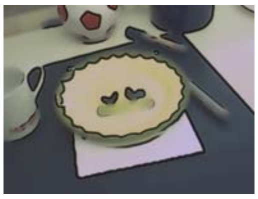
[3]Note that while the full image subtended 43.6 by 33.4 degrees, the experimental object subtended a substantially smaller angle.

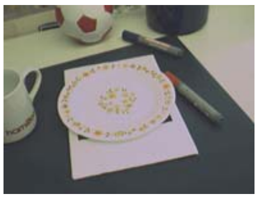
[4]The example object was not used during the main experiment.

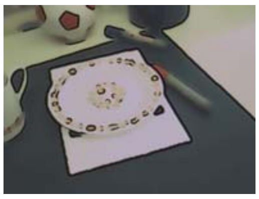

(a) Physical cup, conventional

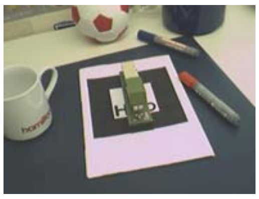

(b) Physical cup, stylized

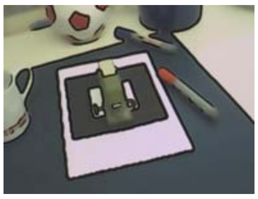

(c) Physical plate, conventional

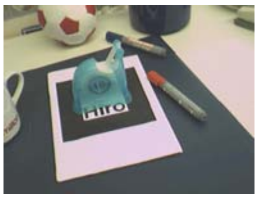

(d) Physical plate, stylized

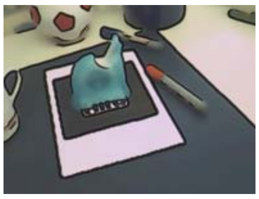

(e) Physical plate, conventional

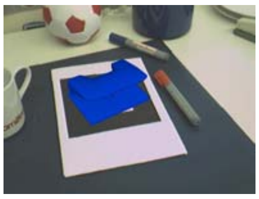

(f) Physical plate, stylized

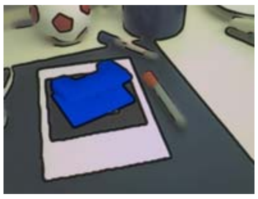

(g) Physical stapler, conventional

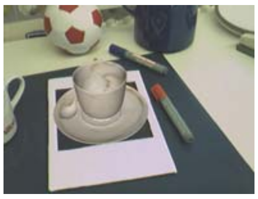

(h) Physical stapler, stylized

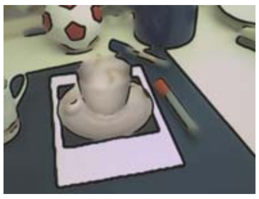

(i) Physical scotch tape, conventional

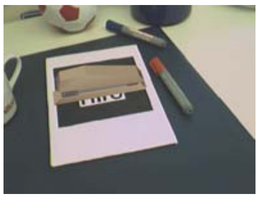

(j) Physical scotch tape, stylized

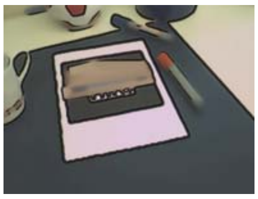

(k) Virtual puncher, conventional

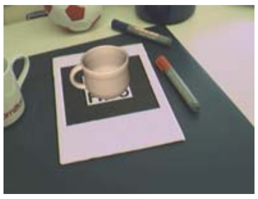

(l) Virtual puncher, stylized

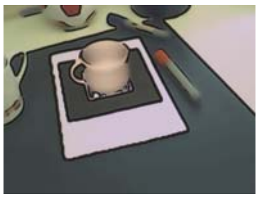

(m) Virtual cup, conventional

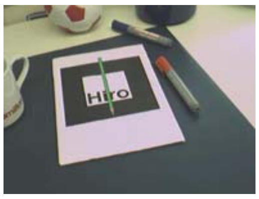

(n) Virtual cup, stylized


(o) Virtual stapler, conventional


(p) Virtual stapler, stylized


(q) Virtual teacup, conventional


(r) Virtual teacup, stylized


(s) Virtual pencil, conventional

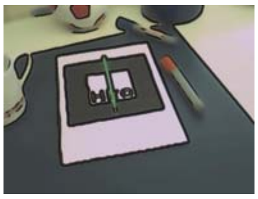

(t) Virtual pencil, stylized

Figure 3.38: Some of the objects shown in the psychophysical study.

Each participant saw all 60 trials in a different random order. Each trial began when the participant pressed a key and ended when they entered their answer. For the Dynamic group, the video sequence was shown in a continuous loop with a 250 ms blank screen between repetitions. For the Static group, the first frame of the recording was shown on the screen. The accuracy and speed of the participants' answers were recorded and separately subjected to a repeated measures analysis of variance (ANOVA), with *AR rendering style* (conventional versus stylized) and *object type* (physical versus virtual) as within-subjects factors and *motion type* (static versus dynamic) as a between-subjects factor. The data from one participant in the Static group were not analyzed, as the participant did not follow the instructions.

## 3.5.3 Results

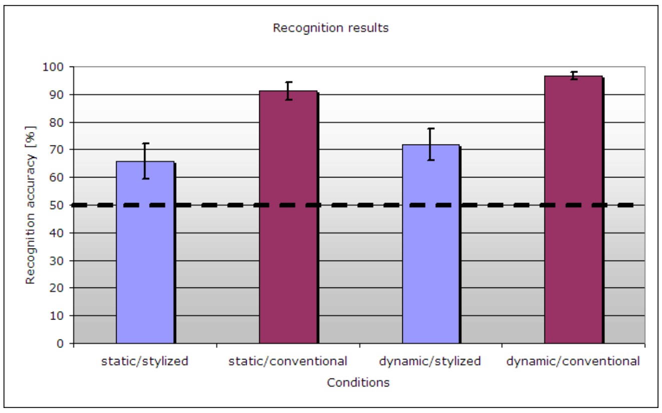
In the following, the statistical analysis of the experimental results is described with respect to both recognition accuracy and reaction times. In this section, several results of the statistical ANOVA analysis are reported. These results are stated in the form F($x$,$y$)=$f$, p<$p$. In this formulation, $x$ and $y$ are parameters describing the experimental setup. They are indirectly derived from the number of groups (here: 2) and the total number of participants (here: 17 without the invalid responses). The actual result of the analysis is represented by the values $f$ and $p$. Expressed in simple terms, a large value of $f$ (significantly larger than 1.0) and an associated small value of $p$ (typically smaller than 0.05) are taken as evidence that a particular experimental condition (e.g., *AR rendering style* or *object type*) influences the recognition accuracy. Conditions which are found to be of no importance for the outcome of the experiment are denoted as *not significant*. A detailed discussion of the ANOVA method can for instance be found in [61, 65, 187].

### Accuracy

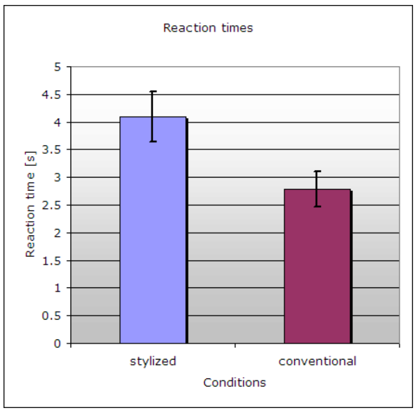
Participants found it significantly harder to tell the difference between physical and virtual objects in stylized AR than in conventionally rendered images (69% versus 94% accuracy ratings, respectively). This is reflected in the significant main effect for *AR rendering style* (F(1,15)=57.345, p<0.0001). The fact that the overall accuracy for stylized AR images is still significantly above chance suggests that while the stylization helped to mask the difference between physical and virtual, it did not completely eliminate it[5]. The fact that the accuracy rate is so low, however, clearly demonstrates the general effectiveness of using stylization. Future experiments with either stronger stylization or different stylization algorithms should easily be able to completely mask the difference between physical and virtual objects.

The results are remarkably consistent between the Static and Dynamic groups (see the graph in Fig. 3.39a). This is reflected in both the lack of an effect of *motion type* (F(1,15)=2.154, p>0.16, *not significant*) and the lack of an interaction between *motion type* and *AR rendering style* (F(1,15)=0.008, *not significant*). This clearly shows that stylization for dynamic sequences did not introduce any artifacts that might have resulted in easier detection of virtual versus physical objects. On the contrary, it seemed that the slight jitters introduced by the sometimes imperfect camera tracking were effectively masked out by the stylization.

[5]Chance level accuracy is what one would expect if the participants were blindly guessing, for example, if they were completely unable to tell the difference between physical and virtual objects. For this experiment, which used a two alternative forced-choice task, chance level is 50%.



(a) Recognition accuracy in percent correct. The dashed line indicates chance level.



(b) Reaction times in seconds.

Figure 3.39: Recognition accuracy and reaction times measured in the study. Error bars in both graphs represent standard error of the mean.

The main effect for *object type* was close to being significant (F(1,15)=3.875, p<0.068): Overall, participants had more difficulty identifiying physical objects than identifying virtual objects (77% versus 86% correct, respectively). This difference is driven almost completely by the physical objects shown as stylized AR images: In the conventionally rendered scenes, virtual and physical objects were correctly labeled 92% and 96% of the time, respectively. In the stylized AR images, however, virtual and physical objects were correctly labeled 80% and 58% of the time, respectively. This trend, which is reflected in the significant interaction between *object type* and *AR rendering style* (F(1,15)=6.985, p<0.02), strongly suggests that most errors are due to the incorrect labeling of physical objects in the stylized images. This result shows that the stylization technique is particularly successful in making physical objects almost indistinguishable from virtual objects.

The effect that virtual objects were relatively more accurately identified compared to physical objects in the stylized AR images is probably caused by the geometric models used in the study. Some of these virtual models are of a rather low graphical quality. Several of the virtual objects have rather unrealistic colors (e.g., almost completely black), and most of them are uniformly colored. Another example of an easily identifiable virtual object is shown in Figure 3.40. The spout of the teapot shown in these images is incorrectly modeled, so that it is displayed as translucent. This was probably a strong hint to most participants that this object is not a physical teapot. Nevertheless, the fact that even in this condition, participants showed a clear decrease in performance testifies to the validity of using stylized AR to blur the distinctions beween the virtual and the real world.

### Reaction Times

The only statistically significant effect that could be found in the reaction times data was that people are slower when assessing stylized AR images than when assessing conventionally rendered AR images (4.0 seconds versus 2.8 seconds, F(1,15)=17.059, p<0.001; see Fig. 3.39b). No other main effects or interactions reached significance. In particular, no difference between the Static and Dynamic conditions was found, showing that dynamic information did not help to speed up the task. Most importantly, virtual and real objects were processed equally fast in

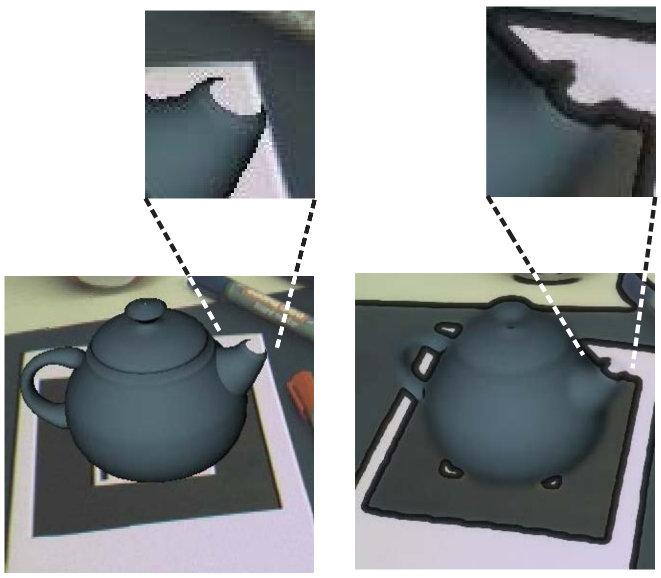


Figure 3.40: Graphical error in the virtual model of a teapot, which was used in the study. Due to incorrect backface culling, the spout of the teapot is only partially displayed. This made it easier for participants to identify the teapot as a virtual object, even in stylized AR (right image).

all conditions suggesting that partipants used the same strategies for both object types. Finally, the difference between stylized and conventionally rendered images reflects the fact that participants had to make a more difficult decision in case of stylized images. This, however, does *not* mean that stylization in itself would result in a less effective AR environment, but rather points towards the difficulty of the task.

### 3.5.4 Conclusion

The results of the psychophysical study showed that presenting the scenes in a stylized manner successfully reduced the detectable differences between physical and virtual objects. It has to be noted that the results were nearly identical for the Static and Dynamic groups, suggesting that the results should generalize to an interactive version of the task in which dynamic information should play an even more important role. Finally, the majority of the errors consisted of falsely believing that some of the physical objects presented in stylized AR were actually virtual objects. This highlights the success of the stylization algorithm in generating a consistent representation of virtual and physical image elements.

As mentioned in Section 3.5.3, some of the virtual object models used in the study were of a rather low graphical quality. It can be assumed that a study based on better graphical models would yield an even more decreased recognition accuracy.

The problem presented to the participants of the described psychophysical study was relatively simple. However, while it could be argued that the experimental setup used in the study was very different from most real-life AR systems, the results of the study indicate that stylized AR is effective. It has been shown that in stylized AR images and video sequences, it is significantly more difficult to distinguish physical objects from virtual objects (and vice versa). This means that a novel user experience is created by applying stylization algorithms in AR, and it also points to an improved immersion in the augmented scene.

## 3.6 Summary

In this chapter, a novel approach to combining real and virtual image elements in augmented reality was presented. The adapted levels of realism in the camera image and the graphical objects in stylized AR result in a novel experience for the user. This way, a unique and impressive augmented reality environment is produced, and possibly a better feeling of immersion is created.

Experiments with the various types of stylization, as well as the psychophysical study described in Section 3.5, have shown that the stylization techniques presented here successfully make the two layers of an AR image - background image and rendered objects - look similar. For some augmented reality scenes, users are genuinely unable to distinguish virtual elements from real objects. In particular this is the case for scenes containing virtual objects which have a scale corresponding to the real world. A good example for such scenes are the virtual models of dishes used in this chapter (e.g., see Fig. 3.13h, 3.25d and 3.35c).

In order to achieve the objectives of stylized augmented reality, it is essential that the real background image and the virtual models look similar. This is accomplished by applying stylization techniques to virtual and real scene elements. While it could be argued that this concept is based on effectively degrading the visual quality of the camera image, enough information is preserved to be useful for many applications. For these scenarios, which do not require a high fidelity rendering of the camera image, stylized AR can be considered a realization of the principle of “functional realism” described by Ferwerda [83].

### 3.6.1 Technical Aspects

The cartoon-like and brush stroke stylization methods for augmented video streams described in this chapter are generally successful at generating a coherent ouput video. However, each type of stylization has certain advantages and limitations. Sometimes, unwanted artifacts can also occur.

In Section 3.3, the advanced method for generating cartoon-like stylized AR images was presented. Since this image stylization filter is designed as a post-processing step, it can easily be combined with any augmented reality rendering system. Its GPU-based implementation is fast and delivers real-time frame rates. The presented post-processing algorithm is based on common basic image operations like photometric filtering and edge detection. It was shown that these basic techniques were specifically adapted for the requirements of stylized augmented reality.

The advanced cartoon-like stylization algorithm consists of a color simplification step and an adaptive edge detection method, which are computed on a per-frame basis. They do not take temporal coherence in consecutive images into account. Depending on the illumination conditions in the surroundings and the quality of the video frames delivered by the camera, the parameters of the algorithm need to be adapted in order to obtain a good separation of uniform regions in the image. However, good results are normally generated with a constant parameter set as long as a similar type of augmented scene is viewed. Only if the video acquisition settings of the camera change significantly or if the lighting in the observed environment varies strongly, the algorithm parameters have to be corrected. A nearly constant setup of the camera image filter has empirically proven to deliver acceptable results under most circumstances. However, very large or extremely small color constrasts between regions in the observed scene can cause the algorithm to produce unsatisfactory stylized images.

The drawback of the brush stroke stylization algorithm presented in Section 3.4 is the fact that the design of the camera image filter results in the so-called “shower door effect”. It is caused by the constant positions of the sampling points and the rendered brush strokes. Most painterly rendering systems make an effort to prevent this shower door effect. However, it was found that a constant position of the brush strokes is necessary for the purpose of stylized augmented reality. As mentioned in Section 3.4, early experiments with brush strokes attached to the projected positions of model particles were not successful. In videos generated with such an algorithm, the virtual models were clearly distinguishable from the background image. The development of a brush stroke stylization algorithm for augmented reality without the shower door effect is one important topic for future research.

The stylization algorithms described in this chapter constitute a useful and promising first realization of the concept of stylized augmented reality. Moreover, the application of an illustrative rendering style to augmented reality is presented in Chapter 4. As a future development, more advanced techniques from the fields of artistic and illustrative rendering could be applied to augmented reality images.

### 3.6.2 Potential Applications and Future Work

Among the main topics in future research are psychophysical studies for the examination of the effects of stylized AR. These could include the repetition of a similar study as described in this Section 3.5 in a more realistic AR setting, e.g., by using a head-mounted display for displaying the images and videos. In a later stage, a study could be executed in an interactive, real-time augmented reality system. Finally, more complicated tasks could be designed which have to be performed by participants both in conventional and stylized augmented reality. An example of such an advanced task performance experiment would be the problem of finding a virtual object (or several virtual objects) in a room in conventional and stylized AR.

The study of stylization effects in the context of psychophysics is also one possible application of stylized augmented reality. For instance, the use of the stylization methods described in this chapter for examining their effect on the recognition of facial expressions has been proposed (see [74] for an example of related previous work).

Among other potential applications of stylized augmented reality are entertainment scenarios and art projects. AR games and other types of virtual entertainment could benefit from the novel user experience and the blurred barrier between virtual and real created by the stylization techniques. An obvious application of stylized augmented reality are art projects and installations, which could be based on the artistic representation of an augmented environment. Stylized AR could also be used for training and education settings. For instance, stylized rendering could be used in interactive AR applications in museums and exhibitions. This way, the experience of visitors could be enhanced with additional virtual exhibits and interactive descriptions in a novel way.

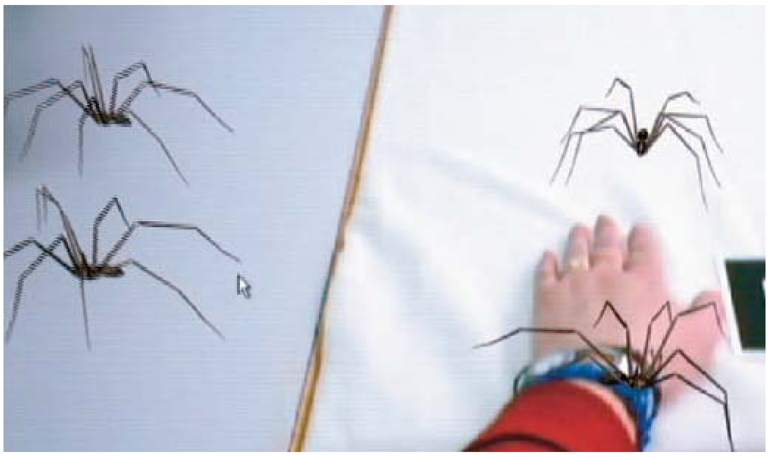


Figure 3.41: Spider phobia treatment in AR (image taken from [118]).

Finally, another possible application for stylized AR are psychotherapy scenarios like phobia therapy. The use of augmented reality for phobia treatment was described for example by Schubert and Regenbrecht [177] and Juan et al. [118]. In the spider phobia application presented by Juan et al., virtual spiders are displayed to the patients, as shown in Figure 3.41. The effect of this treatment in AR, however, might be limited when the patient is able to easily recognize the spider models as virtual objects. It is conceivable that a more effective therapy can be achieved by applying stylized AR. If the patient cannot distinguish virtual spiders from real spiders anymore, the reaction to this virtual stimulus could become stronger.

CHAPTER 4

# Illustrative Visualization

## 4.1 Introduction

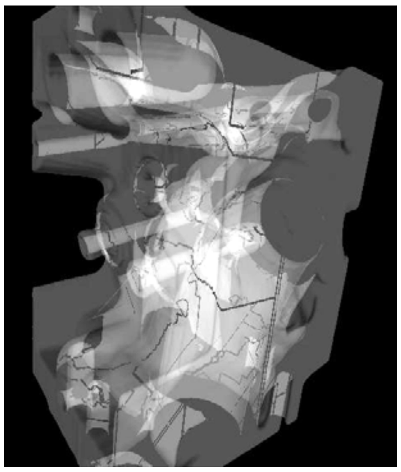
In the preceding chapter, the application of artistic techniques to augmented video streams was discussed. In addition to the use of these basic artistic methods, the development of novel advanced stylized rendering algorithms was a topic pursued in the context of this thesis. Specifically, a new approach to the illustrative visualization of iso-surface datasets and general polygonal models was proposed [12]. This method for illustrative visualization also constitutes the basis for another type of stylized augmented reality, which renders both the real environment and the virtual objects in an illustrative style. The novel illustrative rendering algorithm as well as its application to augmented reality are presented in this chapter.

### 4.1.1 Overview of the New Illustrative Rendering Method

Volumetric datasets have become a widely used format for storing three-dimensional information in application areas like medicine and scientific visualization. The efficient display of such datasets has been a field of active research for many years. One very common method for rendering volume data is the use of polygonal iso-surfaces, which represent boundary surfaces defined by a constant intensity value within the volume. The Marching Cubes algorithm described by Lorensen and Cline is a widespread method for the extraction of such iso-surfaces from a volume dataset [137]. Standard polygon rasterization techniques are then used for displaying the iso-surfaces. This entire process is called indirect volume rendering.

In conventional indirect volume rendering, only the front layer of the iso-surface geometry is visible to the user. Parts of the model which are at a greater depth in the eye coordinate system are suppressed by hidden-surface algorithms like the Z-Buffer. However, the shape of inner structures in a dataset is of great importance for many applications. For instance, inner organ structures or back walls of cavities often are of interest for a physician inspecting an anatomical dataset. In the case of a volume dataset containing a scanned engine block, it might be useful to view hidden mechanical parts or structural weaknesses in non-destructive material testing.

The straightforward solution for displaying hidden structures of an iso-surface is to render the entire geometry with transparent polygons. Parts of the iso-surface geometry which cover the same screen space are combined using standard blending mechanisms (e.g., alpha blending). The drawback of this approach is that the entire polygonal model becomes visible. This can produce a visually complex and difficult to interpret graphical output, if too many structures are shown behind each other (see Fig. 4.1a). Moreover, it is necessary to sort the graphical primitives according to their screen space depth, so that the blending computations yield correct results. This usually leads to a significantly increased computational complexity of the rendering process. The effect of incorrect depth sorting when rendering transparent objects is shown in Figure 4.1b.


(a) Visually complex output due to fully transparent rendering (*Engine* dataset)

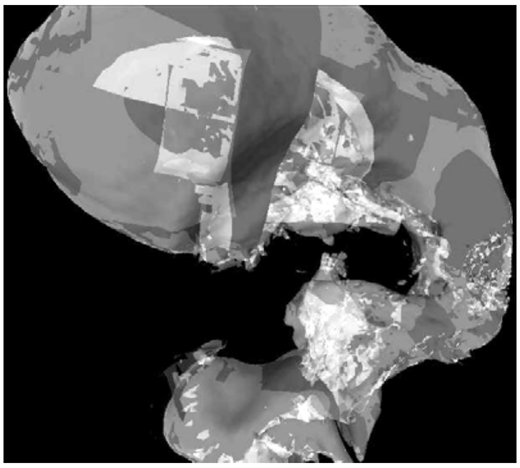

(b) Erratic alpha blending due to incorrect depth sorting (*Ventricle* dataset)

Figure 4.1: Problems of the conventional rendering of transparent structures.

Direct volume rendering, which is not based on precisely defined iso-surfaces, also makes the display of interior structures of an observed dataset possible. However, the necessary definition of a useful transfer function is not trivial [160]. The images generated by direct volume rendering often show many parts of the volume dataset simultaneously, which can make viewing the structures of interest difficult for the user.

Therefore, a novel way of displaying hidden structures of an iso-surface was developed in the context of this thesis. This new algorithm utilizes illustrative rendering methods. The design of the new method aims at achieving the following advantages:

- By using illustrative visualization methods, an easily understandable graphical output is generated. The new algorithm uses silhouette outlines and monochrome hatching for conveying the shape of objects.
- The inner structures of the iso-surface are automatically extracted during the rendering process. In the new approach, the first occluded layer of the iso-surface behind the

front layer is displayed as hidden geometry in a distinctive silhouette-based style. No preprocessing or manual definition of objects is required.

- As an optional addition, the geometry of special inner structures of high interest can be manually specified by the user. This "secondary geometry" always remains visible and is displayed in a special solid style with depth attenuation.

- The algorithm is capable of generating real-time frame rates for most datasets and typical image resolutions.

The design of the new algorithm exploits the programmability of modern graphics processing units. A specialized rendering pipeline was developed, which integrates application-specific object space and image space processing stages. Two geometry rendering passes generate the data for the first and second layer of the iso-surface. In an optional third pass, the information required for displaying the "secondary geometry" is gathered. Finally, an image space processing step combines all the data collected in the previous steps for achieving the desired graphical output. The implementation uses the OpenGL Shading Language (GLSL) [169] and can deliver real-time frame rates in most cases.

In the remainder of this chapter, Section 4.1.2 discusses some related previous work on stylized and illustrative rendering. The new illustrative visualization algorithm is described in Section 4.2. Section 4.3 presents its application to stylized augmented reality. Finally, Section 4.4 concludes this chapter with a summary.

### 4.1.2 Related Work

As mentioned above (see Sec. 3.1.1), artistic and illustrative rendering have been in the focus of active research for several years. The aim of many research trends in non-photorealistic rendering is to imitate painterly or cartoon-like styles in the generated images. An example of painterly rendering is the method of using brush strokes for recreating still images presented by Hertzmann [109], which was later extended by Hertzmann and Perlin for interactively processing video streams [110]. Kaplan et al. have proposed an algorithm for displaying 3D models in different painterly and stroke styles at interactive frame rates [122].

A very important basic principle employed by many stylized rendering techniques is the generation and processing of intermediate image space buffers containing geometric properties of the observed scene. Depth values and transformed normal vectors, which are computed for each image pixel, are frequently used geometric properties. This basic principle was introduced as *G-buffers* by Saito and Takahashi [170]. The new illustrative rendering algorithm presented here is also partly based on such intermediate image space data. Decaudin has described a method for the cartoon-like rendering of 3D objects using depth and normals buffers [77]. Mitchell et al. have presented a GPU-based technique for the extraction of object outlines based on image space information [147]. Nienhaus and Döllner use G-buffers to create different illustrative styles [152]. A framework for mapping G-buffer concepts to modern graphics processing units was described by Eißele et al. [81].

Several researchers have proposed to use colors for conveying the shape of objects or their material properties. Examples include the work of Gooch et al. on stylized lighting for automatic illustration [98], as well as Lum and Ma's watercolor inspired method for rendering surfaces [139]. The new approach described here uses discrete, user-defined colors in order to distinguish the front layer of the iso-surface from the second layer and for highlighting the "secondary geometry".

Monochrome hatching and halftoning are often used in non-photorealistic rendering. These methods map a continuous range of intensities to monochrome representations based on regular patterns or artistic drawing styles. Interrante et al. have discussed the use of a texture containing discrete strokes for conveying the shape of an iso-surface [117, 125]. Hertzmann and Zorin have described methods for the line-art rendering of smooth surfaces [111]. Praun et al. have proposed *tonal art maps* as a primitive for generating an artistic monochrome hatching for 3D objects [164]. The utilization of configurable and programmable graphics hardware for real-time hatching and halftoning has been addressed by several researchers [91, 203]. Secord et al. have presented a method for the high-quality distribution of monochrome drawing primitives based on a probability density function derived from an input image [180]. An application of hatching styles to 3D scans of real world enviroments was developed by Xu and Chen [210]. Strothotte and Schlechtweg describe a method for procedural halftoning, which is used by the algorithm presented in this chapter [189].

Some researchers have addressed the problem of visualizing hidden components of a graphical model. Nooruddin and Turk have described a preprocessing method for classifying interior and exterior parts of a polygonal dataset [154]. Diepstraten et al. have presented an algorithm for displaying hidden structures in technical illustrations using transparency [78] and cutaway techniques [79]. An algorithm for the illustrative display of polygonal models using depth peeling has been described by Nienhaus and Döllner [153]. However, their method extracts consecutive layers of geometry for the entire model, again introducing a certain degree of visual complexity. Moreover, their algorithm typically delivers less than interactive frame rates, whereas the new method described here can generate images in real-time for most datasets.

Illustrative techniques have also been applied to direct volume rendering in order to emphasize structures of interest. Rheingans and Ebert have proposed the *volume illustration* approach for enhancing important features in a volume dataset [168]. A technique that uses stippling for the visualization of volume data has been presented by Lu et al. [138]. Viola et al. describe an automatic method for cutting away irrelevant parts of a volume which occlude significant structures [198]. Yuan and Chen have introduced a system which combines point-based illustrative rendering with direct volume visualization [212]. A method for the real-time generation of line drawings from volume data has recently been presented by Burns et al. [67].

## 4.2 Illustrative Display of Hidden Iso-Surface Structures

The new illustrative rendering algorithm consists of a pipeline of geometry drawing passes followed by an image space processing step. At first, the polygonal geometry of the iso-surface is rendered twice in order to extract the first and second layers of the model. Subsequently, the optional secondary geometry is rendered, if it has been specified by the user. Finally, an image processing shader combines all data collected in the previous stages to achieve the desired illustrative output. This is the main step of the method. In the following, the set of polygons comprising the iso-surface are denoted as **P** and the secondary geometry as **S**.

Figure 4.2 shows an overview of the method. After each of the geometry rendering passes, generated image space data like fragment depth, normal vectors, and computed intensity are gathered. Note that the second and third rendering step take information from previous passes

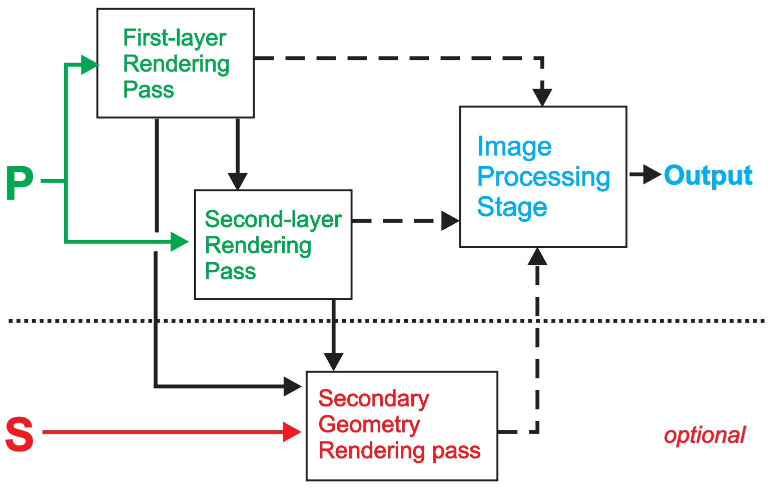
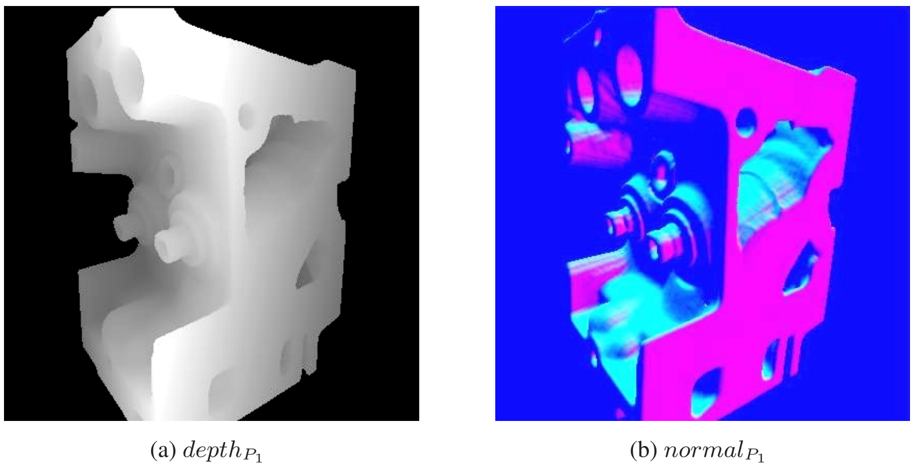

First-layer Rendering Pass
Image Processing Stage
Output
P
Second-layer Rendering Pass
Secondary Geometry Rendering pass
S
optional

Figure 4.2: Overview of the algorithm for the illustrative rendering of hidden iso-surface structures. Solid arrows indicate data that is fed into the geometry rendering passes, dotted arrows data fed into the image processing stage. **P** and **S** stand for the primary iso-surface and the secondary geometry, respectively.

into account for their own computations. This is necessary because the fragment depths of the polygons drawn by the preceding stages are required for correctly rendering the geometry data in subsequent passes.

### 4.2.1 Iso-Surface Rendering Passes

In the first pass of the algorithm, the polygons **P** of the iso-surface are rendered. The standard Z-Buffer test is applied, so that the final image contains the front layer of the iso-surface geometry. For the generation of the image, a special shader is used, which computes the interpolated and normalized normal vectors for each pixel instead of color values. The depth values $depth_{P_1}$ and normal vectors $normal_{P_1}$ are the image space output of this pass and can be accessed by the following steps of the algorithm. Figure 4.3 shows the output generated by the first rendering pass for a view of the *Engine* dataset.



(a) $depth_{P_1}$ (b) $normal_{P_1}$

Figure 4.3: Image space data generated for the first layer of the *Engine* dataset. The components of the normal vectors are represented as RGB-values in Fig. 4.3b.

In the next step of the rendering pipeline, the iso-surface polyons **P** are drawn once again. This time, the aim is to generate the image that results from removing the first layer of the geometry. The depth information from the first pass is used to achieve this effect using a technique called depth peeling [153]. Each fragment rasterized in this rendering step has to pass two depth tests. In the first test, the depth of the currently regarded polygon fragment is compared to the depth of the first-pass geometry at the same location. The new fragment has to be at a greater depth than the value stored in $depth_{P_1}$, otherwise it is culled. The result of this test is that the first layer of the iso-surface geometry is suppressed, or “peeled away”. Subsequently, each fragment has to pass the standard Z-Buffer test so that a coherent second-layer image is generated. This process is illustrated in Algorithm 5.

**Algorithm 5** Pseudocode demonstrating the principle of depth peeling.

```
for all polygons $p \in \mathbf{P}$ do
   $F$ := rasterize $p$;
   for all fragments $f \in F$ do
      // Depth peeling test
      if ($f$.depth <= $depth_{P_1}$($f$.x,$f$.y)) then
         continue; // Skip this fragment
      endif
      // Standard Z-Buffer test
      if ($f$.depth > $depth_{P_2}$($f$.x,$f$.y)) then
         continue; // Skip this fragment
      endif
      $depth_{P_2}$($f$.x,$f$.y) := $f$.depth;
      $normal_{P_2}$($f$.x,$f$.y) := $f$.normal;
   done
done
```

Only fragments which pass both tests contribute to the image space output of the second rendering stage. The diagram in Figure 4.4 demonstrates the depth peeling technique. As shown here, the second rendering pass yields the first layer of geometry behind the directly visible polyons.

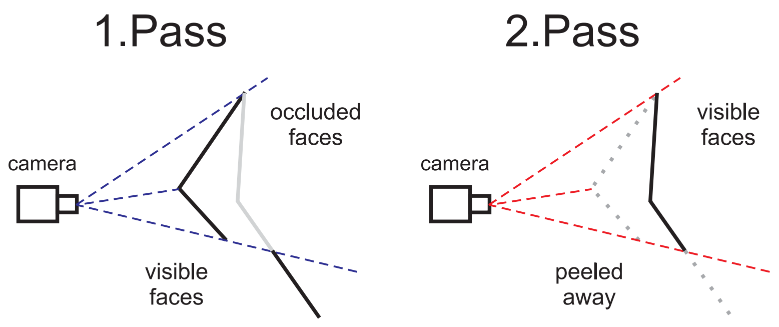


Figure 4.4: Illustration of the depth peeling technique (adapted from [153]). Thick black lines indicate polygons which are visible in the image generated by the respective rendering stage.

The output of the second rendering pass again consists of depth and normal information, $depth_{P_2}$ and $normal_{P_2}$. Figure 4.5 depicts the data computed for the second layer of the *Engine* volume dataset.

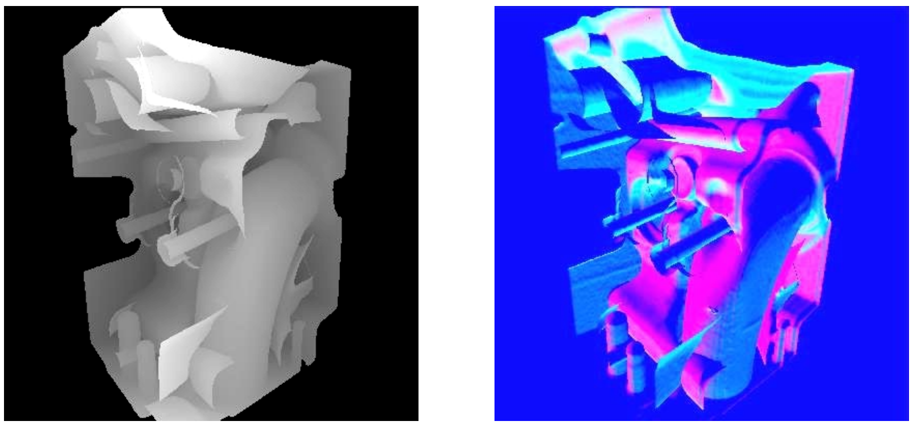

(a) $depth_{P_2}$ (b) $normal_{P_2}$

Figure 4.5: Image space data generated for the second layer of the *Engine* dataset.

### 4.2.2 Optional Rendering of Secondary Geometry

If a polygonal dataset containing secondary geometry has been specified by the user, the optional third geometry rendering pass is performed. In contrast to the two initial iso-surface rendering steps, this pass generates an intensity texture. The computed intensities are later used by the image processing stage as brightness values for the secondary geometry pixels. All polygons in $\mathbf{S}$ are rasterized. Initially, the intensity of each fragment is calculated using diffuse reflection:

$$I_S^0(x,y) = max(normal_S(x,y) \cdot \mathbf{lightDir}, 0) \tag{4.1}$$

As shown in Equation 4.1, the initial intensity value $I_S^0$ is the result of the dot product of the normal vector of the fragment, $normal_S(x,y)$, and the user-defined light direction $\mathbf{lightDir}$. A special depth attenuation scheme is then applied to the fragment intensities. The aim of this process is to make the depth relationships in the generated image better understandable. Although the secondary geometry is supposed to remain always visible, a comprehensible representation of its distance relative to iso-surface structures is desired. Therefore, the fragment depth of the secondary geometry is compared to the depth values retrieved from the first two rendering passes.

$$I_S(x,y) = I_S^0(x,y) \cdot \begin{cases} 1, & depth_S(x,y) \; < depth_{P_1}(x,y) \\ \alpha, & depth_{P_1}(x,y) \leq depth_S(x,y) < depth_{P_2}(x,y) \\ \beta, & depth_{P_2}(x,y) \leq depth_S(x,y) \end{cases}$$

$$\text{with } 0 < \beta < \alpha < 1 \tag{4.2}$$

The depth of the secondary geometry fragments is denoted as $depth_S$ in Equation 4.2. Depth values computed during the primary rendering passes are contained in $depth_{P_1}$ and $depth_{P_2}$, as described in Section 4.2.1. The following rule determines the attenuation of the initial fragment intensity $I_S^0$: Secondary geometry fragments which are directly visible, i.e., not behind any iso-surface layer, retain their full intensity. Fragments which would be occluded by the front layer are attenuated with factor $\alpha$, those which are also behind the second iso-surface layer with factor $\beta$. These factors have to be in the interval $[0; 1]$ and are parameters of the algorithm.

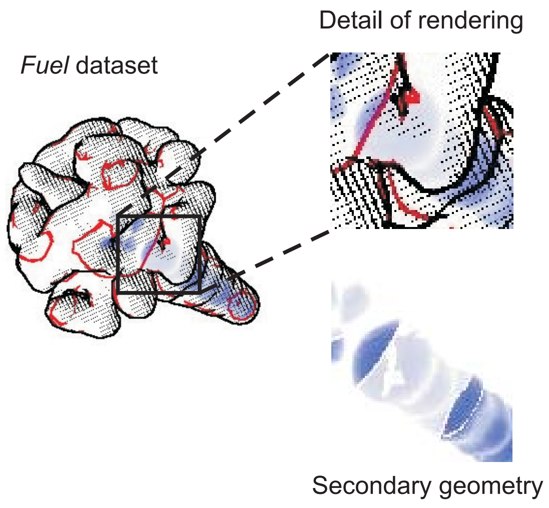


Figure 4.6: Increasing attenuation of secondary geometry pixels which are behind one or two iso-surface layers, respectively.

This depth attenuation method helps the user understand the spatial relationship between the secondary geometry and the iso-surface. Figure 4.6 illustrates this effect. In the central part of the image detail, the secondary geometry is shown with less intensity because here it is occluded by two iso-surface layers. The output of the optional third rendering step is the final intensity value $I_S$, which is accessed by the image processing stage. Moreover, the depth of the secondary geometry fragments, $depth_S$, is also stored for use by the final step of the rendering pipeline.

### 4.2.3 Image Processing Stage

All of the aforementioned geometry rendering passes generate images with the full resolution of the OpenGL output window. The size of this OpenGL viewport is determined by the user. In the final stage of the algorithm, all the previously generated data are combined using an image processing step. This step is performed by drawing a textured rectangle, which again has the same size as the OpenGL window. The intermediate images generated by the geometry rendering passes are used as input textures for a special shader program. In this shader, a number of image processing tasks are performed, which eventually yield the desired output rendering.

#### Silhouette Detection

As mentioned above, the display of silhouettes is a central feature of the illustrative rendering method presented here. Silhouettes are detected in image space for both the first and the second layer of the iso-surface. The silhouette detection method used here is similar to the approach described by Mitchell et al. [147]. Discontinuities in the depth and normal vector images are the basis for computing the response of the silhouette detection filter.

The gradient magnitude is computed for the depth maps of both iso-surface layers, $depth_{P_1}$ and $depth_{P_2}$. In order to determine the gradient magnitude, partial derivatives $\frac{\partial\, depth_{P_n}}{\partial x}$ and $\frac{\partial\, depth_{P_n}}{\partial y}$ ($n \in \{1, 2\}$) are obtained using the Sobel edge detection filter. The norms of the resulting gradient vectors, $|\nabla depth_{P_1}|$ and $|\nabla depth_{P_2}|$, indicate discontinuities in the depth images. A binary response is determined for the depth discontinuity filter based on a user-defined threshold, $depthThresh$, as shown in Equation 4.3.

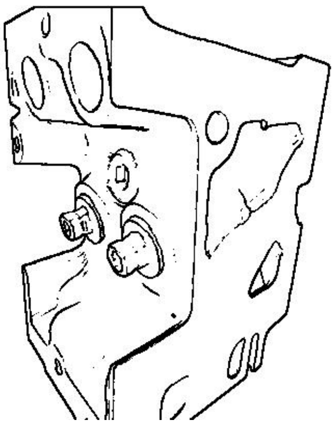
$$depthResp_{P_n}(x,y) = \begin{cases} 0, & |\nabla depth_{P_n}|(x,y) < depthThresh \\ 1, & |\nabla depth_{P_n}|(x,y) \geq depthThresh \end{cases} \tag{4.3}$$

For finding discontinuities in the normal images, the normal vector stored in each pixel is compared to its four direct neighbours. This comparison is computed as a dot product between the corresponding vectors. As shown in Equation 4.4, the four resulting dot products are added up in order to obtain a normal vector similarity value $normalSim_{P_n}$. These computations are performed for both layers of the iso-surface, yielding $normalSim_{P_1}$ and $normalSim_{P_2}$, respectively. A smaller normal similarity value indicates a significant discontinuity.

$$\begin{aligned} normalSim_{P_n}(x,y) \; = \; & normal_{P_n}(x,y) \cdot normal_{P_n}(x+1,y) + \\ & normal_{P_n}(x,y) \cdot normal_{P_n}(x-1,y) + \\ & normal_{P_n}(x,y) \cdot normal_{P_n}(x,y+1) + \\ & normal_{P_n}(x,y) \cdot normal_{P_n}(x,y-1) \end{aligned} \tag{4.4}$$

$$normalResp_{P_n}(x,y) = \begin{cases} 0, & normalSim_{P_n}(x,y) \geq normalThresh \\ 1, & normalSim_{P_n}(x,y) < normalThresh \end{cases} \tag{4.5}$$

A binary normal discontinuity response is computed by comparing the similarity value to a user-defined threshold (see Equation 4.5). The greater the threshold value $normalThresh$ is, the more pixels contribute to normal map silhouettes. Finally, an overall silhouette detection response is determined for each iso-surface layer. Depth as well as normal discontinuities are taken into account by calculating the logical OR of both response types, i.e., $silhouette_{P_n}(x,y) = depthResp_{P_n}(x,y) \vee normalResp_{P_n}(x,y)$. The output of the silhouette detection process for both layers of the *Engine* dataset is shown in Figure 4.7.


(a) $silhouette_{P_1}$

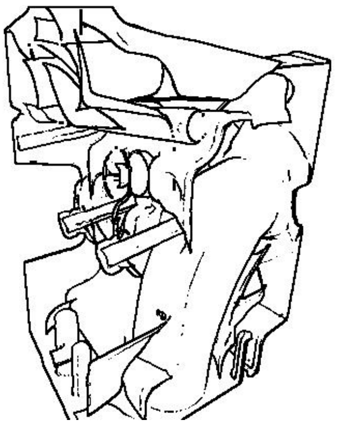

(b) $silhouette_{P_2}$

Figure 4.7: Silhouette detection responses computed for the first and second layer of the *Engine* dataset iso-surface.

### Procedural Halftoning

In the illustrative rendering style described here, the shape of the front layer of the iso-surface is not only represented with silhouettes. In order to make a better understanding of the geometry of the iso-surface possible, and to achieve a clear distinction between first-layer and second-layer structures, diffuse reflection intensities are added to the graphical output. The intensity of each first-layer fragment is computed as the dot product of a user-defined light direction and the stored normal vector. As shown in Equation 4.6, the aforementioned light direction vector **lightDir** is again used for this calculation.

$$I_{P_1}(x,y) = max(normal_{P_1}(x,y) \cdot \mathbf{lightDir}, 0) \tag{4.6}$$

The main aim of the rendering algorithm is the display of hidden structures behind the first iso-surface layer. It is therefore necessary to show the diffuse reflection intensities with a method which allows second-layer silhouettes to remain visible. A monochrome halftoning method is used for recreating a black-and-white cross-hatching style found in technical illustrations. The procedural screening approach described by Strothotte and Schlechtweg [189] is utilized, which does not require additional input textures for describing the hatching pattern.

The coordinates of each fragment which is to be dithered are first mapped to dither coordinates $(s,t)$. This is done using a mapping function $\mathbf{M}$:

$$\begin{aligned}(x',y') &= R_\theta \cdot (x,y)\\ (s,t) = \mathbf{M}(x',y') &= \left(\frac{x' \ mod \ n}{n}, \frac{y' \ mod \ n}{n}\right)\end{aligned} \tag{4.7}$$

In Equation 4.7, the screen-space coordinates of the currently regarded fragment are denoted as $(x,y)$. These coordinates are first rotated by the angle $\theta$, which is selected by the user. This is done by computing the product of the rotation matrix $R_\theta \in \Re^{2x2}$ and the input coordinate vector. Due to this multiplication, the resulting halftoning pattern will be rotated relative to the screen space coordinate axes. This creates a more natural look of the black-and-white hatching image. Subsequently, the rotated fragment coordinates are mapped to the real number range $[0;1]$ using the mapping function $\mathbf{M}$. The dither coordinates $(s,t)$ are the basis for determining whether the fragment is displayed as an opaque black pixel or as translucent. The choice of the user-defined variable $n$ in Equation 4.7 determines the size in pixels of the hatching pattern.

$$\tau(s,t) = \begin{cases} c_{cross} \cdot t & s \leq c_{cross} \\ (1-c_{cross})s + c_{cross} & s > c_{cross} \end{cases} \tag{4.8}$$

$$I_{hatching}(x,y) = \begin{cases} 0, & I_{P_1}(x,y) < \tau(s,t) \\ 1, & I_{P_1}(x,y) \geq \tau(s,t) \end{cases} \tag{4.9}$$

For each pixel, a halftoning threshold $\tau$ is computed as a function of the dither coordinates $(s,t)$. The calculation of $\tau$ depends on the parameter $c_{cross}$ (see Equation 4.8). $c_{cross}$ determines the minimum intensity necessary for generating perpendicular cross-hatching strokes in addition to the parallel strokes which are the basis for the halftoning pattern. As described in

Equation 4.9, the diffuse reflection intensity $I_{P_1}(x, y)$ is then compared to the local threshold $\tau(s, t)$. If the intensity is large enough, the output value $I_{hatching}(x, y)$ will be one, otherwise zero. The black-and-white pattern computed by this halftoning method for the intensity range $[0; 1]$ is depicted in Figure 4.8.

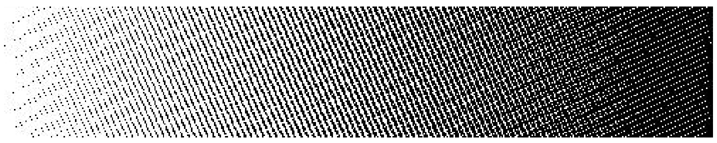


Figure 4.8: Black-and-white pattern generated by the halftoning method for the continuous intensity range $[0; 1]$.

### Display of Secondary Geometry

In the final output image, the secondary geometry is displayed as a solid polygonal object with a special color. The brightness of secondary geometry fragments is determined by the intensity value $I_S(x, y)$, which has been computed in the optional third geometry rendering pass (see Section 4.2.2).

Before the final color for each pixel in the output image is computed, a specific depth test is performed for the secondary geometry. In places where second-layer silhouette pixels have been detected, only such secondary geometry fragments are to be displayed which are in front of the second layer of the iso-surface.

$$I'_S(x, y) = I_S(x, y) \cdot \begin{cases} 0, & silhouette_{P_2}(x, y) \wedge (depth_{P_2}(x, y) \leq depth_S(x, y)) \\ 1, & otherwise \end{cases} \tag{4.10}$$

Equation 4.10 shows the additional depth test for secondary geometry pixels, which yields the final intensity value $I'_S$. Due to this test, the depth relationships between second-layer silhouettes and the secondary geometry are correctly represented.

### Computation of Final Pixel Color

In order to generate the final output image, all the data computed for each pixel so far are combined. The image is initialized with a user-defined background color called $paperColor$. Typically, a bright background color is used to create the look of a technical illustration on paper. The second-layer silhouettes are then drawn over the background. They are displayed with the color $backLayerColor$ in all places where $silhouette_{P_2}$ indicates a silhouette fragment. Subsequently, the secondary geometry is blended over the background according to its computed intensity $I'_S$. The color of the secondary geometry is also selected by the user and stored in the variable $secGeomColor$. Finally, the front-layer of the iso-surface is taken into account. Every pixel which has a first-layer silhouette response of one or a hatching intensity of zero is displayed black. In the current implementation of the algorithm, black is always used for the first-layer geometry, but it could easily be replaced with any other color. The piece of pseudocode shown in Algorithm 6 summarizes the process of determining the final pixel color. This code uses a terminology similar to the functionality of modern shading languages.

**Algorithm 6** Computation of final pixel color.

```
vec3 pixelColor;
pixelColor = mix($paperColor$, $backLayerColor$, $silhouette_{P_2}(x,y)$);
pixelColor = mix($pixelColor$, $secGeomColor$, $I'_S(x,y)$);
pixelColor *= (1.0 - $silhouette_{P_1}(x,y)$);
pixelColor *= $I_{hatching}(x,y)$;
```

In the pseudocode in Algorithm 6, the variable `pixelColor` is an RGB color vector, which will store the final output color of the pixel at position $(x, y)$. In the code, the linear blend of two colors is computed by the function `mix`, i.e., `mix`($c_1$, $c_2$, $\alpha$) = $(1-\alpha)c_1 + \alpha c_2$. The black color of the first-layer geometry is generated by multiplying the components of `pixelColor` with a scalar value of zero, indicated by an `*=` operator.

### 4.2.4 Implementation Details

The capability for achieving real-time frame rates has been one of the main objectives of the design and the implementation of the algorithm. The new illustrative visualization method was realized with the OpenGL Shading Language [169], which was used to implement all of the aforementioned shader programs. The entire approach is executed on the graphics processing unit (GPU) and does not require any preprocessing on the CPU.

Each type of intermediate image-space information (e.g., $depth_{P_n}$, $normal_{P_n}$, $depth_S$ etc.) is stored in a separate texture image in onboard memory. The normal and intensity images are generated as the components of standard RGBA textures, while $depth_{P_n}$ and $depth_S$ have a special depth texture format provided by OpenGL. In each geometry rendering pass, the intermediate data are written into the framebuffer. Subsequently, they are copied into texture images which are accessed by later stages of the algorithm. This is done with the `glCopy-TexSubImage2D()` function.

In the final step of the algorithm, a rectangle covering the entire OpenGL window is drawn using the image processing shader. The image processing shader reads the intermediate data by accessing the corresponding texture images. In particular for the silhouette detection step, a large number of texture accesses is required. For each fragment, a neighborhood of five or nine texels has to be read for the normal and depth textures, respectively. In order to avoid a loss of performance due to the repeated calculation of texel addresses, a special precomputation scheme is used. This texel address precomputation method is similar to the one described in Section 3.3.4.

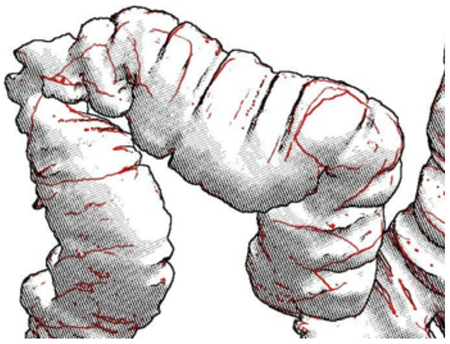
One significant advantage of the design of the rendering pipeline (see Fig. 4.2) is the fact that all image processing tasks are performed in the same shader progam. This way, no redundant texture accesses are necessary. All the previously computed data relevant for the current pixel are loaded once from the intermediate textures and can be used for several computations. One example is the first-layer normal information, $normal_{P_1}$, which is needed for the silhouette detection and the calculation of the diffuse reflection intensity.

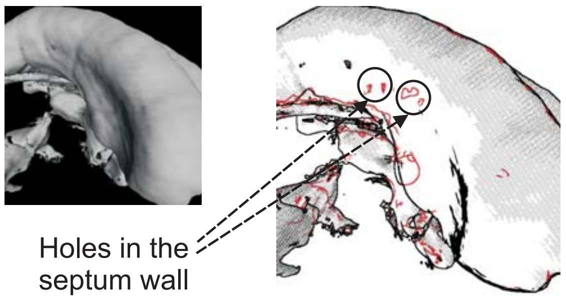
Many of the computations in the implementation of the rendering pipeline use standard functions provided by the shading language. Examples include `step()`, which performs a boundary check, and `mix()`, which computes a linear blending of vectors. These functions are normally executed efficiently on the GPU. Due to the extensive use of these functions, the implementation requires almost no conditional branches and no loop statements on the GPU, which are notoriously slow.

### 4.2.5 Results

The algorithm was tested with a number of example datasets. Most of them are polygon meshes representing iso-surfaces generated from volume data. For some of the datasets, secondary geometry was created by extracting a second surface with a different iso-value. Example images rendered with the new method are shown in Figure 4.10. The *Engine* dataset (Fig. 4.10a) is a CT scan of two cylinders of an engine block. The result of a simulation of diesel injection into a combustion chamber is contained in the *Fuel* dataset (Fig. 4.10b). In this case, high-intensity voxels are displayed as secondary geometry. The *NegHip* example (Fig. 4.10c) is a simulation of the spatial probability of the electrons in a protein molecule. It also contains secondary geometry extracted from high-intensity voxels. The *Screwdriver* dataset (Fig. 4.10d) is the detailed mechanical design of an electric screwdriver, with some inner parts highlighted as secondary geometry. Unlike the other examples shown here, the *Screwdriver* dataset is not an iso-surface extracted from volume data, but a manually created CAD model. A CT scan of a human colon is shown in the *Colon* example, and the *Ventricle* dataset is an MRI scan of the ventricular system of a human brain (Fig. 4.10e and 4.10f).

The example images and animations generated with the new illustrative rendering algorithm show that the method can convey the shape of hidden structures of the iso-surface. This is also illustrated in Figure 4.9a, which shows a detail of the colon dataset, where the shape of the back wall of the colon is suggested by second-layer silhouettes. In Figure 4.9b, holes in the septum of the patient are visible in the *Ventricle* dataset. These are defects caused by a degenerative process, and they are significant for medical diagnosis and treatment. Figure 4.11 illustrates how manually specified secondary geometry is highlighted by the algorithm in the case of the *Screwdriver* dataset.

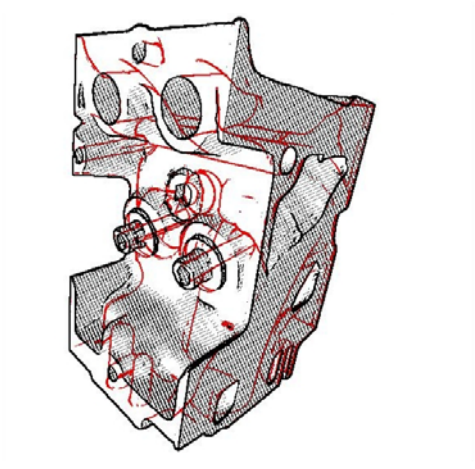

(a) Close-up of a section of the *Colon* dataset

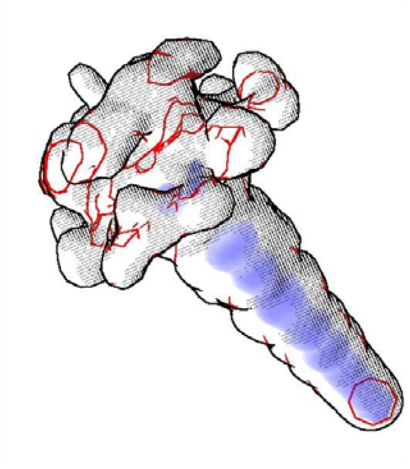


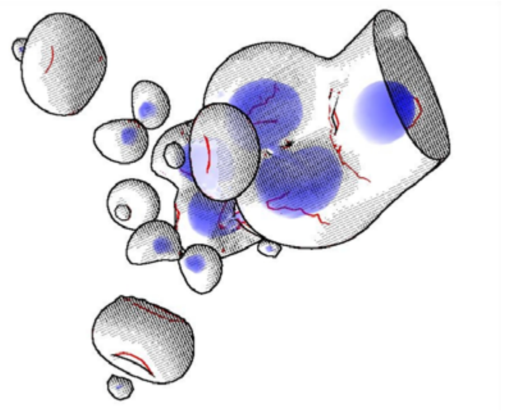
(b) A significant detail in the *Ventricle* dataset displayed by the algorithm

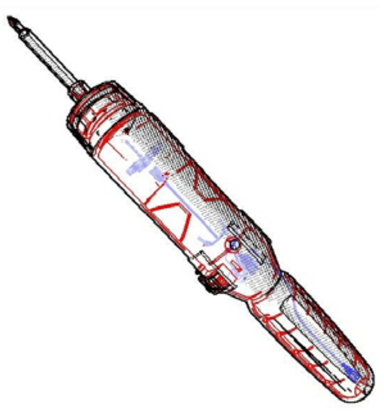
Figure 4.9: Details of two renderings generated with the algorithm.

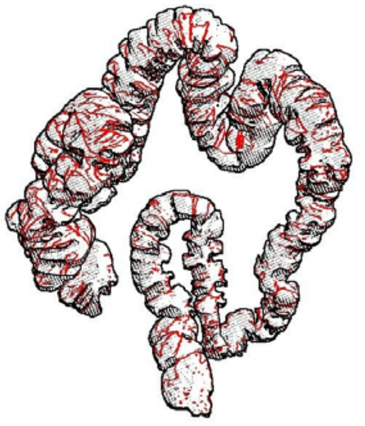
A number of tests were performed for measuring the frame rates achieved with the illustrative visualization method. These benchmarks were run on a computer with a Pentium 4 processor running at 2.8 GHz using a graphics card with an NVidia GeForce FX 6600 GT chipset. Animation sequences with a length of at least 1000 frames were used to compute the average performance for each test run. Table 4.1 shows a comparison of frame rates measured with standard rendering and with the illustrative method for the *Engine* dataset.


(a) *Engine* dataset


(b) *Fuel* dataset


(c) *NegHip* dataset


(d) *Screwdriver* dataset


(e) *Colon* dataset

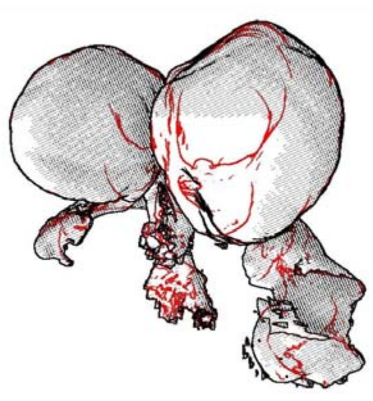

(f) *Ventricle* dataset

Figure 4.10: Example images generated with the illustrative rendering algorithm. (Parameters used for all images: $\alpha = 0.7$, $\beta = 0.21$, $normalThresh = 3.98$, $depthThresh = 0.001$, $\theta = 20.0$, $n = 4.0$, $c_{cross} = 0.9$)

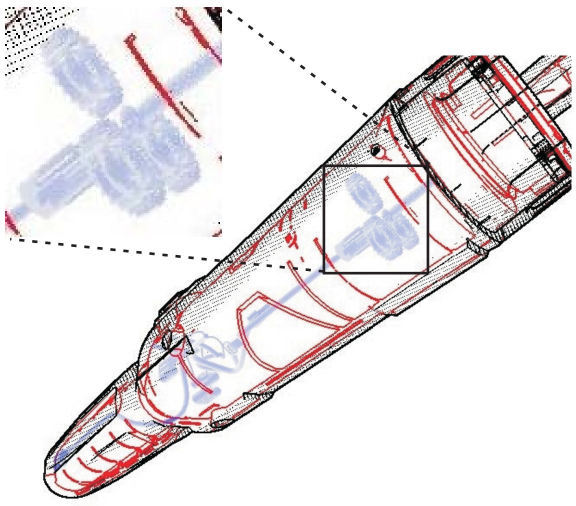


Figure 4.11: Detailed polygonal model of an electric screwdriver. Some inner parts are used as secondary geometry (see enlarged detail).

Table 4.1: Comparison of frame rates for standard and illustrative rendering of the *Engine* dataset (311k vertices) at different resolutions.

| | **1024x768** | **800x600** | **640x480** |
|---|---|---|---|
| **standard** | 96.01 | 99.05 | 100.85 |
| **illustrative** | 21.33 | 28.10 | 34.83 |

In Table 4.2, rendering speeds achieved by the algorithm for the other five test datasets are listed. Benchmark runs were performed for different output resolutions. As illustrated in both benchmark tables, the performance of the method depends on two main factors. The first is the size of the iso-surface mesh. A large number of polygons slows down the image generation because of the multiple geometry rendering passes. Moreover, the frame rate is influenced strongly by the output resolution. A larger output window increases the size of the intermediate textures and the number of image processing operations, resulting in a reduced rendering speed. Still, the algorithm is capable of delivering real-time performance in most cases. Even for large datasets and image resolutions, frame rates of close to or above 20 fps have been measured.

Table 4.2: Frame rates measured for the other five datasets.

| **Dataset** | **#vertices** | **1024x768** | **800x600** | **640x480** |
|---|---|---|---|---|
| Colon | 1,076k | 18.16 | 22.96 | 27.17 |
| Screwdriver | 487k | 26.65 | 40.69 | 58.07 |
| Ventricle | 201k | 31.00 | 45.94 | 61.98 |
| NegHip | 17k | 30.96 | 50.41 | 76.78 |
| Fuel | 6.4k | 32.29 | 52.30 | 81.49 |

## 4.3 Illustrative Augmented Reality

Using the visual style of the illustrative visualization method presented in the previous section, an additional type of stylized augmented reality was realized in the context of this thesis. In this illustrative augmented reality system, the real camera image is filtered so that it recreates the same look of a technical illustration on paper which is generated by the illustrative visualization algorithm. An example image rendered by the illustrative AR system is shown in Figure 4.12.

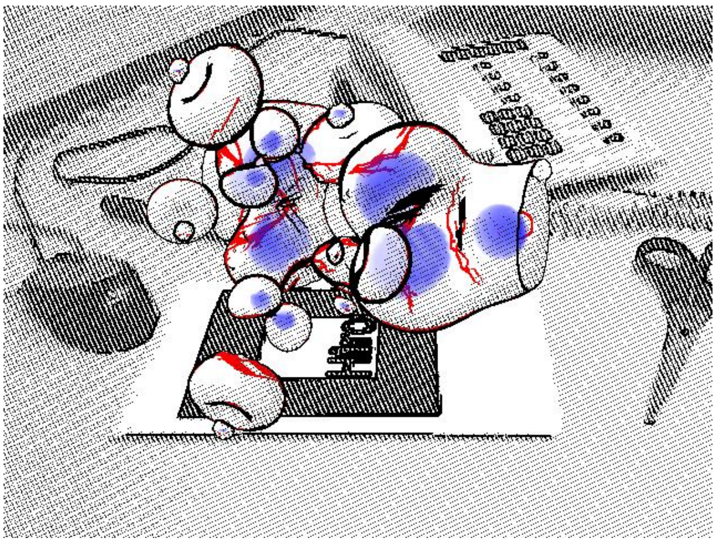


Figure 4.12: Image generated by the illustrative augmented reality system.

### 4.3.1 Description of the Illustrative AR System

In the illustrative augmented reality system, the camera image and the virtual objects which constitute the augmented scene are handled separately. This concept is similar to the design of the original cartoon-like stylized AR system (see Sec. 3.2) and the brush stroke stylization method for augmented images (see Sec. 3.4). However, unlike in those types of stylized AR, the camera image processing step is also performed on the graphics processing unit (GPU) in illustrative augmented reality. Figure 4.13 shows an overview of the method.

As illustrated in Figure 4.13, the original camera image is processed in a separate step. In this processing stage, edges in the camera image are detected, and a low-pass filter is applied to it. The data generated in this step are then used by the final rendering stage, which is a modified and enhanced version of the image processing step of the illustrative visualization algorithm (see Sec. 4.2.3).

#### Camera Image Filter

At the beginning of the illustrative AR pipeline, the original camera image is copied from the frame buffer into a texture image in graphics card memory. This texture is then used as input texture for an image filtering step. The camera image filter is executed by rendering a

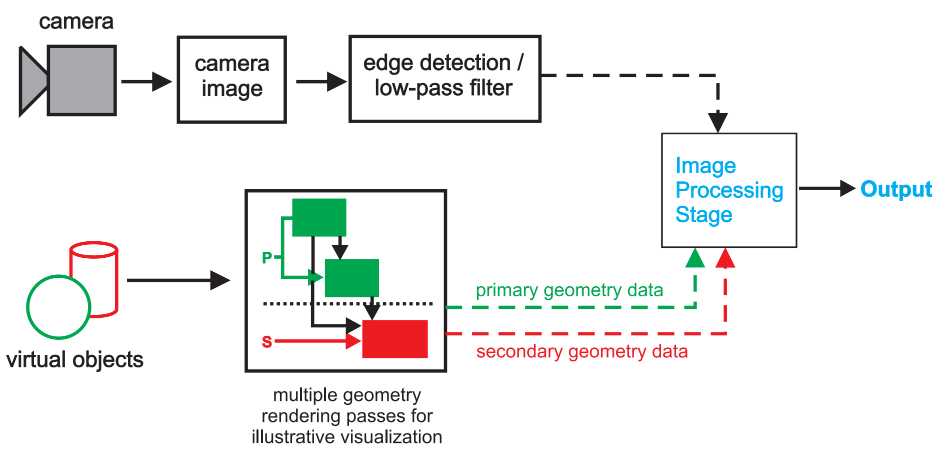


Figure 4.13: Overview of the illustrative augmented reality rendering pipeline. In this diagram, dotted arrows represent image space data which is fed into the final image processing stage.

two-dimensional rectangle at the full resolution of the original image. During this rendering process, a specific image processing shader program is activated.

Two tasks are performed by the camera image filter. The most important aspect is the detection of edges in the camera image. An edge detection strategy is used which is similar to the method applied in the GPU-based cartoon-like stylization filter described in Section 3.3.3. Each camera image pixel is converted into the YUV color space. Partial derivatives are then computed for each color channel using the Sobel filter, yielding the gradient magnitudes $(|\nabla Y|, |\nabla U|, |\nabla V|)$. Like in the GPU-based cartoon-like stylization filter, these gradient magnitudes are then used in an adaptive edge detection process based on color as well as intensity constrasts. As shown in Equation 4.11, the user-defined parameter $\alpha \in [0;1]$ determines the relative influence of color contrasts (in the U and V channels) and intensity contrasts (in the Y channel) when computing an edge detection response. The computed edge detection response $edge_{cam}$ is then compared with the threshold $camEdgeThresh$ in order to obtain the binary silhouette flag $silhouette_{cam}$ for the camera image (see Eq. 4.12).

$$edge_{cam} = (1-\alpha) \cdot |\nabla Y| + \alpha \cdot \frac{|\nabla U| + |\nabla V|}{2} \tag{4.11}$$

$$silhouette_{cam} = \begin{cases} 0, & edge_{cam} < camEdgeThresh \\ 1, & otherwise \end{cases} \tag{4.12}$$

As the second output of the camera image filter, the low-pass filtered pixel intensity is computed. A box filter is applied to the image intensities in a 3 x 3 pixel neighborhood. The final output of the camera image filtering step is the determined silhouette flag, $silhouette_{cam}$, as well as the filtered intensity, $I_{cam}$, for each pixel. These data are again stored in a separate texture image, which is later used as additional input for the final image composition stage.

## Final Image Composition

The image composition step of the illustrative AR system is a modified version of the final image processing step used in the original illustrative visualization algorithm (see Sec. 4.2.3).

This new image composition stage, however, computes two possible color values for each pixel in the output image. The first computed alternative is the output color which would be generated by the illustrative visualization method, i.e., `pixelColor`. Additionally, a representation of the camera image in an illustrative style is also rendered for every pixel location. This second color alternative is denoted as `camPixelColor`.

The computation of `camPixelColor` consists of three steps. At the beginning of the process, the pixel color is initialized with the background color, `paperColor` (cf. Sec. 4.2.3). Subsequently, the color of the stylized camera image pixel is set to black if an input image silhouette was detected at this location. Finally, in order to achieve a look for the output image which visually corresponds to the desired illustrative style, a monochrome hatching is generated based on the pixel intensity. The same hatching technique is used as in the illustrative visualization system (see Sec. 4.2.3). Since the halftoning threshold $\tau$ has already been computed in the image processing shader, only a comparison with the camera image intensity at this location is required in order to generate the hatching. This comparison is shown in Equation 4.13. The parameter $\gamma$ is introduced into the comparison in order to make an overall brightening or darkening of the stylized camera image possible. This way, a similar brightness can be generated for the camera image as for the virtual objects. As the last operation in the computation of the camera pixel color, the hatching is then taken into consideration. If a hatching value of zero was determined, the output pixel is also set to black. Algorithm 7 summarizes the computation of the stylized camera image pixel. Here again, a terminology similar to the functionality of modern shading languages is used.

$$I_{camHatching}(x,y) = \begin{cases} 0, & \gamma \cdot I_{cam}(x,y) < \tau(s,t) \\ 1, & \gamma \cdot I_{cam}(x,y) \geq \tau(s,t) \end{cases} \tag{4.13}$$

**Algorithm 7** Computation of the color for the stylized camera image pixels.

```
vec3 camPixelColor;
camPixelColor = paperColor;
camPixelColor *= (1.0 - $silhouette_{cam}(x,y)$);
camPixelColor *= $I_{camHatching}(x,y)$
```

After the computation of the stylized camera image as well as the output of the original illustrative algorithm, the image composition step selects which of the two color alternatives is used as the final result. In places which are covered by the geometry of the virtual objects, their illustrative representation is shown. Otherwise, the stylized camera image pixel is rendered. In order to determine which pixels are covered by the virtual objects, the first geometry rendering pass (see Sec. 4.2.1) is adapted so that it generates a flag for every pixel which is modified when drawing the polygons of the virtual objects. This flag is stored in the fourth component of the RGBA output texture, which otherwise only holds the computed normal vectors.

While it might appear that the calculation of two color alternatives is a suboptimal computation strategy, the generation of the stylized camera image pixel requires very little computational effort (see Algorithm 7). The selection of the correct color value can then efficiently be done with a single call to the `mix()` function provided by the shading language. The alternative solution of using conditional processing based on the virtual object flag would introduce an inefficient `if`-statement into the image composition step. Such an inefficient conditional branching on the GPU is avoided by the described rendering strategy used in the illustrative AR system.

### 4.3.2 Results

The illustrative augmented reality system was tested with several example scenes. Figure 4.14 shows three test datasets rendered in the illustrative AR environment. The *Submarine* dataset is the polygonal model of a small submarine (see Fig. 4.14a). The *DC10* model (Fig. 4.14b) is the graphical representation of a jet airplane. Finally, the *NegHip* example (Fig. 4.14c) again is the iso-surface extracted from a volumetric dataset containing the spatial probability of the electrons in a molecule (this example was also used in Sec. 4.2.5). For the *Submarine* and *NegHip* models, certain inner structures were explicitly specified as secondary geometry.

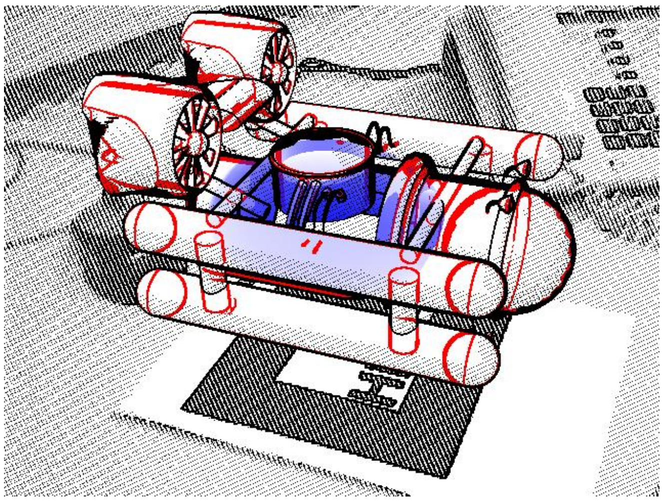
As demonstrated in Figure 4.14, the illustrative AR system generates augmented images in which real and virtual scene elements are rendered in a similar style. Since the same type of hatching and silhouette representation is used, a good visual similarity is achieved. Unlike in the previously described stylized AR approaches, however, certain visual differences are inherently created in the rendering process. Both the second-layer silhouettes and the secondary geometry of the virtual object are displayed in different colors, while the stylized camera image is strictly black-and-white. Moreover, the secondary geometry is rendered as a solid object, distinguishing it from the other objects in the image. Still, the stylization filter applied to the camera image creates a very uniform look in the generated output video frames.

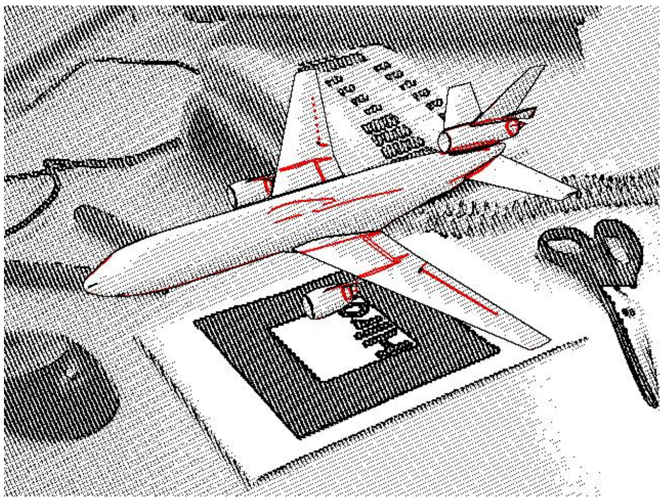
Table 4.3 lists average frame rates measured for the three test datasets in the illustrative augmented reality system. For each example case, an interactive test session with a duration of at least 480 frames was performed. These benchmarks were measured on a computer with an Intel Pentium 4 Xeon processor running at 2.66 GHz and a graphics card based on an NVidia GeForce FX 6800 GT chipset. As mentioned earlier, the augmented reality system uses a Firewire webcam delivering a resolution of 640 x 480 pixels.

The results shown in Table 4.3 demonstrate that the illustrative AR method is capable of delivering a video stream at a sustained rate of 30 fps for every example scene. The reasons for this high performance are twofold. On the one hand, the original illustrative visualization algorithm is designed to deliver real-time frame rates for image sizes significantly higher than the webcam resolution. For a resolution of 640 x 480 pixels, the illustrative rendering method itself achieves 60 fps or more unless the polygonal model is very complex (see Table 4.2). Moreover, the camera image filter used in the illustrative AR system is relatively simple, in particular because it is a single-pass method. It can therefore be executed significantly faster than the rather complex stylization filter of the GPU-based cartoon-like AR system (see Sec. 3.3). The overall performance of the illustrative AR system in the examined test cases was only limited by the speed of the webcam, which acquires images at a rate of approximately 30 Hertz.

Table 4.3: Frame rates measured for the test datasets in the illustrative AR system.

| Dataset | #vertices | Frame Rate | Rendering Time |
|---:|---:|:---:|:---:|
| | | (fps) | (msecs) |
| Submarine | 482k | 30.38 | 17 |
| NegHip | 17k | 30.36 | 12 |
| DC10 | 7k | 30.73 | 12 |


(a) *Submarine* model


(b) *DC10* model

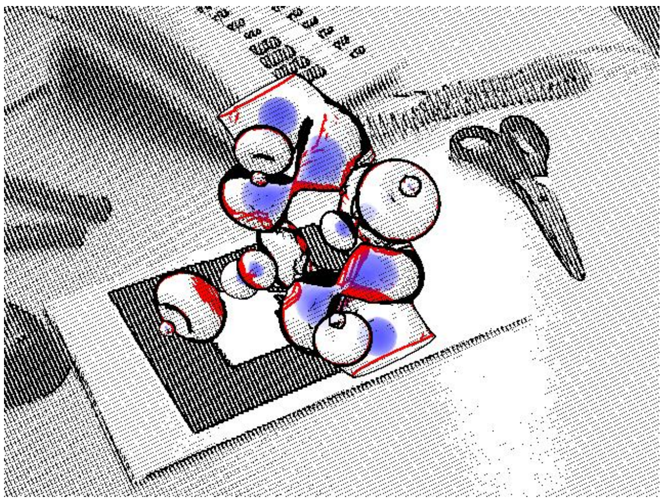

(c) *NegHip* dataset

Figure 4.14: Example images generated with the illustrative AR system.

## 4.4 Summary

In this chapter, a novel method for the illustrative rendering of iso-surfaces was presented. The new algorithm utilizes depth peeling, silhouette extraction, and monochrome hatching and combines them in an efficient way in a single rendering pipeline. The generated images make a simultaneous inspection of the outer surface as well as inner structures possible. Due to the selected style of rendering, spatial relationships and the shape of hidden objects can easily be understood.

One major advantage of the illustrative visualization algorithm is the lack of any preprocessing of the input data. Any polygonal dataset can be loaded into the system and then be displayed in the new illustrative style. The depth peeling step automatically extracts the second-layer geometry. Since it is always assumed that the second iso-surface layer contains the relevant hidden structures, difficulties can arise in the case of more complex datasets. If areas of interest are occluded by several layers of polygons, they have to be manually specified as secondary geometry in order to remain visible. However, the continual display of such secondary structures is integrated efficiently into the rendering pipeline.

Although the performance of the illustrative visualization algorithm depends on the size of the displayed dataset and the image resolution, it was empirically found that it is capable of achieving real-time frame rates in most cases.

The application of the illustrative visualization style to augmented reality constitutes another type of stylized AR. As discussed in Chapter 3, any type of stylization can be used for realizing a stylized augmented reality system as long as real and virtual objects obtain a similar look. Although the illustrative AR approach creates a slightly different appearance for the graphical models in the scene, a sufficient visual similarity is achieved. The augmented environment is rendered in a style recreating the look of a technical illustration on paper. Thanks to the illustrative visualization algorithm, inner structures and secondary geometry of the graphical models are displayed in an easily comprehensible way. Education and training as well as (collaborative) design tasks could be among the potential applications for the illustrative augmented reality approach.

CHAPTER 5

# Conclusions and Future Work

## 5.1 Conclusions

This thesis described a number of new developments in the field of augmented reality. The main focus in most of the projects presented here was to find new and advanced ways of rendering augmented video streams. Considering that much of the existing research in augmented reality deals with the problems of tracking, user interaction, display technologies, and system design, this thesis put a different emphasis on the relatively underrepresented topic of specialized graphics algorithms. In the vast majority of existing AR systems, standard rendering methods (e.g., straightforward OpenGL calls) are used for displaying the virtual objects. While this approach is sufficient for some applications, especially when only a limited amount of mostly symbolic augmentations is shown, the combination of artificial-looking computer graphics with the fully detailed real camera image can lead to a dissatisfactory user experience. The novel AR rendering methods presented in this thesis have attracted a rather large amount of attention, and they hopefully manage to inspire more research on the topic of specialized display algorithms for augmented video streams. The main conclusion of this thesis can be considered to be the proposal that **the development of specialized rendering algorithms be regarded as a research problem of equal importance as the other main technical challenges of augmented reality**.

Two different main research trends were described in this thesis. The topic of medical augmented reality was discussed in Chapter 2. Chapters 3 and 4 presented the fusion of stylized rendering and AR. In the medical augmented reality chapter, an application-specific method for handling static occlusion in the augmented scene was described. This approach improves the visual realism of the generated output video and represents an advanced way of rendering AR images. The sections on stylized and illustrative augmented reality introduced the new idea of applying stylization techniques to both real and virtual scene elements, which also constitutes a new way of rendering for AR. In the following, the conclusions for each section are summarized.

**Medical Augmented Reality**. The medical AR framework *ARGUS* was presented, which is built using commercially available and certified medical technology. The *ARGUS* system was the basis for a number of advanced developments in the fields of tracking, user interaction, and the aforementioned occlusion-aware rendering method. Later, the framework also served as the foundation for a novel approach to semiautomatic volume classification, which confirmed its usefulness and stability [6, 128]. While the *ARGUS* system in its current state cannot be considered ready for use in an actual clinical context, its basic design philosophy can probably help to facilitate the transition of medical AR into the practical application.

**Stylized Augmented Reality**. The concept of a stylized AR video stream containing artistic depictions of virtual as well as real scene elements was proposed. Several different implementations of this principle were presented. The described algorithms for creating cartoon-like and brush stroke representations of augmented images are specifically adapted variations of existing stylized rendering techniques or completely new developments. Some of the presented methods consist of separate processing paths for the camera image and the graphical models, using standard graphics library calls and image processing on the CPU. In other cases, the programmability of modern graphics processing units was employed to achieve a fast and high quality postprocessing of image data. All of the presented stylization algorithms are capable of generating an output video stream at interactive or even real-time frame rates. The presented example cases showed that using the different types of AR stylization, augmented images are generated in which real and virtual objects are difficult to distinguish. The decreased discernability of graphical models in stylized augmented reality was also confirmed by the psychophysical study performed in the context of this thesis. Consequently, a novel experience for the user is created, and possibly a better feeling of immersion. The stylized AR concept could be the basis for a number of possible applications, e.g., art installations in the form of an "Augmented Painting" and video games, as well as a number of other uses like psychotherapy applications and further studies in psychophysics.

**Illustrative Visualization**. A new method for the illustrative display of hidden structures in iso-surfaces and general polygonal models was presented. The generated output images recreate the visual style of a technical illustration on paper. It was shown that the algorithm can produce useful and aesthetically pleasing renderings of datasets ranging from anatomical volume data to polygonal CAD designs. Since the algorithm is implemented using shader programs for modern programmable graphics processing units, it can generate images in real-time even for complex models and high image resolutions. No manual preprocessing of the datasets is necessary because the method automatically extracts the second layer of geometry. Moreover, additional inner structures can optionally be specified by the user. Although the automatic extraction of second-layer geometry does not always produce the structures of interest in a dataset, and although certain rendering artifacts are sometimes generated by the underlying basic techniques, the new method has proven to be capable of rendering useful images in most cases. Based on this illustrative visualization method, an additional type of stylized AR system was realized. In illustrative augmented reality, both the camera image and the virtual objects resemble the look of a technical illustration. Although here, as in the other realizations of stylized AR, the fidelity of the rendered camera image is reduced, the illustrative AR approach could be used for applications in training, education, and design.

## 5.2 Future Directions of Research

The main topic in future research on stylized augmented reality is the development of improved and advanced stylization algorithms. The generation of artistic or illustrative representations of augmented video streams poses a unique set of challenges. Only a monoscopic camera image stream at a relatively low resolution together with estimated poses is available as input data. This data stream has to be processed on a per-frame basis because the output is used in an interactive system (i.e., it is not possible to apply multi-pass video processing methods). Moreover, the system has to generate real-time frame rates, and ways of integrating the virtual information have to be investigated. One of the most promising directions for future developments is the exploitation of temporal coherence in consecutive camera frames. More advanced existing artistic and illustrative rendering algorithms can be applied to augmented video streams in order to create a better visual quality or new visual styles. Finally, the incorporation of new types of input data into the rendering process could be investigated. These could include (semi-)static models of the real environment or geometry data from a partial 3D reconstruction based on the camera images.

There are several possible routes for enhancing and improving the presented illustrative visualization method. A more sophisticated technique for automatically finding inner structures of interest would be one important improvement of the algorithm. As another possible research topic, the illustrative visualization pipeline could be redesigned to be more configurable. In an advanced illustrative visualization system, the number of extracted geometry layers and the display style applied for each layer could be selected by the user. In order to maintain the real-time characteristics of the approach, an intelligent rendering strategy with an optimized automatic configuration of the visualization pipeline would be required for such a development. Such an advanced system could even incorporate the use of dynamically generated shader program code.

Finally, the exploration of potential applications for the combination of stylization techniques and augmented reality is one of the most important areas in future research. As an obvious example, an AR game could be created, which is based on the principle of stylized augmented reality. With regard to applications in electronic entertainment, a special emphasis could be put on using widespread hardware platforms as basis for the implementation, for instance commercial game consoles with camera add-ons or modern cell phones with built-in cameras. Moreover, an interactive art installation could be prepared in order to demonstrate the concept in public. Other possible scenarious include training and education, e.g., by using stylized AR in an augmented museum exibition or in the context of cultural heritage applications. As another possibility, the use of stylization techniques in augmented environments designed for psychotherapy and psychophysics could be investigated. The aforementioned phobia therapy application is one example for such an augmented reality system which could benefit from the effects of stylization. The various stylization techniques could also be used to support experimental perception research in psychophysics. An example for the application of stylization methods in psychophysics would be a possible study on the recognition of virtual and real faces in stylized videos. Lastly, advanced artistic stylization methods for real scenes with added virtual objects could in the long term also be used in applications beyond interactive AR systems. The automatic generation of high quality stylized augmented video streams might be of interest for the TV and movie industries, for example.

# Authored and Co-Authored Publications

## Publications in Conference Proceedings and Journals

[1] D. Bartz, D. Mayer, J. Fischer, S. Ley, A. del Río, S. Thust, C. Heussel, H. Kauczor, and W. Straßer. Hybrid Segmentation and Exploration of the Human Lungs. In *Proc. of IEEE Visualization*, pages 177–184, October 2003.

[2] D. Bartz, B. Schnaidt, J. Cernik, L. Gauckler, J. Fischer, and A. del Río. Volumetric High Dynamic Range Windowing for Better Data Representation. In *Proc. of ACM International Conference on Virtual Reality, Computer Graphics, Visualization and Interaction in Africa (Afrigraph)*, pages 137–144, January 2006.

[3] D. Bartz and J. Fischer and A. del Río and J. Hoffmann and D. Freudenstein. VIRTUE: A Navigated Virtual Endoscopy System for Maxillo-Facial and Neurosurgery. In *Proc. of 3D Modelling*, April 2003.

[4] A. del Río, J. Fischer, D. Bartz, and W. Straßer. Fast Rendering of Large Encoded Isosurfaces from Uniform Grid Datasets. In *Proc. of Vision, Modeling, and Visualization*, pages 71–78, November 2005.

[5] A. del Río, J. Fischer, M. Köbele, D. Bartz, J. Hoffmann, F. Duffner, M. Tatagiba, and W. Straßer. Intuitive Volume Classification in Medical AR. In *Jahrestagung der Deutschen Gesellschaft für Computer-und Roboterassistierte Chirurgie e.V. (CURAC)*, September 2005.

[6] A. del Río, J. Fischer, M. Köbele, D. Bartz, and W. Straßer. Augmented Reality Interaction for Semiautomatic Volume Classification. In *Proc. of Eurographics Symposium on Virtual Environments*, pages 113–120, October 2005.

[7] J. Fischer. *Rendering Methods for Augmented Reality*. Ph.d. dissertation, University of Tübingen, June 2006.

[8] J. Fischer, D. Bartz, A. del Río, Z. Salah, J. Orman, D. Freudenstein, J. Hoffmann, and W. Straßer. VIRTUE: Ein System der navigierten virtuellen Endoskopie für die MKG- und Neurochirurgie (Poster). In *Jahrestagung der Deutschen Gesellschaft für Biomedizinische Technik*, pages 498–499, 2003.

[9] J. Fischer, D. Bartz, M. Neff, C. Westendorff, J. Hoffmann, D. Freudenstein, F. Duffner, M. Tatagiba, and W. Straßer. Flexible In-Situ Interaction for Medical AR with the ARGUS System (poster). In *Jahrestagung der Deutschen Gesellschaft für Computer- und Roboterassistierte Chirurgie e.V. (CURAC)*, September 2005.

[10] J. Fischer, D. Bartz, and W. Straßer. Occlusion Handling for Medical Augmented Reality using a Volumetric Phantom Model. In *Proc. of ACM Symposium on Virtual Reality Software and Technology*, pages 174–177, November 2004.

[11] J. Fischer, D. Bartz, and W. Straßer. Artistic Reality: Fast Brush Stroke Stylization for Augmented Reality. In *Proc. of ACM Symposium on Virtual Reality Software and Technology*, pages 155–158, November 2005.

[12] J. Fischer, D. Bartz, and W. Straßer. Illustrative Display of Hidden Iso-Surface Structures. In *Proc. of IEEE Visualization*, pages 663–670, October 2005.

[13] J. Fischer, D. Bartz, and W. Straßer. Intuitive and Lightweight User Interaction for Medical Augmented Reality. In *Proc. of Vision, Modeling, and Visualization*, pages 375–382, November 2005.

[14] J. Fischer, D. Bartz, and W. Straßer. Reality Tooning: Fast Non-Photorealism for Augmented Video Streams. In *IEEE and ACM International Symposium on Mixed and Augmented Reality Poster Proceedings*, pages 186–187, October 2005.

[15] J. Fischer, D. Bartz, and W. Straßer. Stylized Augmented Reality for Improved Immersion. In *Proc. of IEEE Virtual Reality*, pages 195–202, March 2005.

[16] J. Fischer, D. Cunningham, D. Bartz, C. Wallraven, H. Bülthoff, and W. Straßer. Measuring the Discernability of Virtual Objects in Conventional and Stylized Augmented Reality. In *Proc. of Eurographics Symposium on Virtual Environments*, pages 53–61, May 2006.

[17] J. Fischer, D. Cunningham, D. Bartz, C. Wallraven, H. Bülthoff, and W. Straßer. Virtual or Real? Judging The Realism of Objects in Stylized Augmented Environments (poster). In *Tübinger Wahrnehmungskonferenz (TWK)*, page 119, March 2006.

[18] J. Fischer and A. del Río. A Fast Method for applying Rigid Transformations to Volume Data. In *Proc. of WSCG*, pages 55–62, February 2004.

[19] J. Fischer, A. del Río, M. Mekić, D. Bartz, J. Hoffmann, and W. Straßer. Effizientes Spiegeln von Volumendaten an einer beliebigen Ebene für die MKG-Chirurgie. In *Jahrestagung der Deutschen Gesellschaft für Computer-und Roboterassistierte Chirurgie e.V. (CURAC)*, November 2003.

[20] J. Fischer, M. Eichler, D. Bartz, and W. Straßer. Model-based Hybrid Tracking for Medical Augmented Reality. In *Proc. of Eurographics Symposium on Virtual Environments*, pages 71–80, May 2006.

[21] J. Fischer, M. Mekić, A. del Río, D. Bartz, and J. Hoffmann. Prototyping a Planning System for Orbital Reconstruction. In *Proc. of Workshop Bildverarbeitung für die Medizin*, pages 264–268, March 2004.

[22] J. Fischer, M. Neff, D. Freudenstein, and D. Bartz. Medical Augmented Reality based on Commercial Image Guided Surgery. In *Proc. of Eurographics Symposium on Virtual Environments*, pages 83–86, June 2004.

[23] J. Fischer, M. Neff, D. Freudenstein, D. Bartz, and W. Straßer. ARGUS: Harnessing Intraoperative Navigation for Augmented Reality. In *Jahrestagung der Deutschen Gesellschaft für Biomedizinische Technik*, pages 42–43, September 2004.

[24] J. Fischer, M. Neff, D. Freudenstein, J. Hoffmann, and D. Bartz. Realtime Exchange of Data with an Image-Guided Surgery System (poster). In *Jahrestagung der Deutschen Gesellschaft für Biomedizinische Technik*, pages 56–57, September 2004.

[25] J. Fischer, M. Neff, D. Freudenstein, J. Hoffmann, F. Duffner, D. Bartz, and W. Straßer. Practical Reality Augmentation for Medicine using Off-the-shelf Medical Equipment. In *Jahrestagung der Deutschen Gesellschaft für Computer-und Roboterassistierte Chirurgie e.V. (CURAC)*, October 2004.

[26] J. Fischer, H. Regenbrecht, and G. Baratoff. Detecting Dynamic Occlusion in front of Static Backgrounds for AR Scenes. In *Proc. of Eurographics Symposium on Virtual Environments*, pages 153–161, May 2003.

[27] D. Freudenstein, J. Fischer, D. Bartz, M. Neff, M. Tatagiba, and F. Duffner. Novel User Interaction Paradigm for Medical Augmented Reality. In *Jahrestagung der Deutschen Gesellschaft für Neurochirurgie*, May 2005.

[28] J. Hoffmann, D. Troitzsch, C. Westendorff, D. Bartz, J. Fischer, A. del Río, F. Dammann, and S. Reinert. Image-Guided Planning and Navigation-Assisted Surgery for Orbital and Orbita-Zygomatic Reconstructions. In *Jahrestagung der Deutschen Gesellschaft für Biomedizinische Technik*, pages 40–41, September 2004.

[29] J. Hoffmann, D. Troitzsch, C. Westendorff, A. del Río, J. Fischer, D. Bartz, and S. Reinert. Bilddatengestützte Navigation und optische Endoskopie zur minimal invasiven kraniomaxillofazialen Chirurgie (Poster). In *Jahrestagung der Deutschen Gesellschaft für Biomedizinische Technik*, pages 504–505, 2003.

[30] J. Hoffmann, C. Westendorff, D. Bartz, J. Fischer, A. del Río, and S. Reinert. Computergestützte Planung und Navigationsgestütztes operatives Vorgehen bei komplexen Orbitawandeffekten. In *Jahrestagung der Deutschen Gesellschaft für Computer-und Roboterassistierte Chirurgie e.V. (CURAC)*, September 2005.

[31] J. Hoffmann, C. Westendorff, D. Bartz, J. Kaminsky, J. Fischer, and S. Reinert. Use of Computer-aided Planning and Image-guided Navigation for Surgical Treatment of Complex Orbital Wall Defects. In S. Weber, F. Langlotz, J. Bier, and T. Lüth, editors, *Computer Aided Surgery around the Head International Symposium*, page 81, 2005.

[32] D. Mayer, D. Bartz, J. Fischer, S. Ley, A. del Río, S. Thust, H. Kauczor, W. Straßer, and C. Heussel. Hybrid Segmentation and Virtual Bronchoscopy based on CT Images. *Academic Radiology*, 11(5):551–565, May 2004.

[33] D. Troitzsch, J. Hoffmann, C. Westendorff, D. Bartz, J. Fischer, F. Dammann, and S. Reinert. Computergestützte virtuelle Planung und navigationsgeführte Reposition von Jochbeinfrakturen. In *Jahrestagung der Deutschen Gesellschaft für Computer-und Roboterassistierte Chirurgie e.V. (CURAC)*, 2004.

[34] C. Westendorff, D. Gülicher, J. Fischer, D. Bartz, A. del Río, S. Reinert, and J. Hoffmann. Computergestützte Planung und CT-Daten-basierte Navigation zur Rekonstruktion des lateralen Mittelgesichts bei Jochbeinfehlstellungen. In *Jahrestagung der Deutschen Gesellschaft für Computer-und Roboterassistierte Chirurgie e.V. (CURAC)*, September 2005.

[35] C. Westendorff, J. Kaminsky, A. del Río, J. Fischer, M. Tatagiba, S. Reinert, and J. Hoffmann. Resektion eines ausgedehnten Keilbeinflühelmeningeoms mit Infiltration der Orbita unter Gebrauch Computergestützter Planung und CAD/CAM Technologie zur Rekonstruktion. In *Jahrestagung der Deutschen Gesellschaft für Computer-und Roboterassistierte Chirurgie e.V. (CURAC)*, September 2005.

## Technical Reports

[36] D. Bartz, B. Schnaidt, J. Cernik, L. Gauckler, J. Fischer, and A. del Río. Volumetric High Dynamic Range Windowing for Better Data Representation. Technical Report WSI-2005-03, Wilhelm Schickard Institute for Computer Science, Graphical-Interactive Systems (WSI/GRIS), University of Tübingen, January 2005.

[37] J. Fischer and D. Bartz. A Pointillism Style for the Non-Photorealistic Display of Augmented Reality Scenes. Technical Report WSI-2005-05, Wilhelm Schickard Institute for Computer Science, Graphical-Interactive Systems (WSI/GRIS), University of Tübingen, May 2005.

[38] J. Fischer and D. Bartz. Real-time Cartoon-like Stylization of AR Video Streams on the GPU. Technical Report WSI-2005-18, Wilhelm Schickard Institute for Computer Science, Graphical-Interactive Systems (WSI/GRIS), University of Tübingen, September 2005.

[39] J. Fischer and D. Bartz. Utilizing Image Guided Surgery for User Interaction in Medical Augmented Reality. Technical Report WSI-2005-04, Wilhelm Schickard Institute for Computer Science, Graphical-Interactive Systems (WSI/GRIS), University of Tübingen, March 2005.

## Patents

[40] J. Fischer and D. Bartz. Vorrichtung und Verfahren zur Steuerung von operationsunterstützenden Informationssystemen (Menüsystemsteuerung für operationsunterstützende Navigationssysteme). Patent pending (German patent office no. 102004049258.1), October 2004.

# Bibliography

[41] K. Agusanto, L. Li, Z. Chuangui, and N. Sing. Photorealistic Rendering for Augmented Reality using Environment Illumination. In *Proc. of IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 208–216, October 2003.

[42] Artcyclopedia. Artists by Movement: Pointillism. http://www.artcyclopedia.com/, 2005.

[43] R. Azuma. A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

[44] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent Advances in Augmented Reality. *IEEE Computer Graphics and Applications*, 21(6):34–47, November/December 2001.

[45] M. Bajura, H. Fuchs, and R. Ohbuchi. Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient. In *Proc. of ACM SIGGRAPH*, pages 203–210, July 1992.

[46] D. Ballard and C. Brown. *Computer Vision*. Prentice-Hall, 1982.

[47] D. Bartz. *Large Model Visualization: Techniques and Applications*. Dissertation, University of Tübingen, May 2001.

[48] D. Bartz. Virtual Endoscopy in Research and Clinical Practice. *Computer Graphics Forum*, 24(1):111–126, March 2005.

[49] D. Bartz, Ö. Gürvit, D. Freudenstein, H. Schiffbauer, and J. Hoffmann. Integration of Navigation, Optical and Virtual Endoscopy in Neurosurgery and Oral and Maxillofacial Surgery. In *Proc. of Caesarium on Computer Aided Surgery*, November 2001.

[50] D. Bartz, J. Hoffmann, D. Freudenstein, Ö. Gürvit, H. Schiffbauer, F. Duffner, and W. Straßer. Integration von Navigation, optischer und virtueller Endoskopie in der Neuro- sowie Mund-, Kiefer- und Gesichtschirurgie. In *Jahrestagung der Deutschen Gesellschaft für Computer-und Roboterassistierte Chirurgie e.V. (CURAC)*, October 2002.

[51] R. Bauernschmitt, M. Feuerstein, E. Schirmbeck, J. Traub, G. Klinker, S. Wildhirt, and R. Lange. Improved Preoperative Planning in Robotic Heart Surgery. In *Proc. of IEEE Computers in Cardiology*, pages 773–776, September 2004.

[52] M.-O. Berger. Resolving Occlusion in Augmented Reality: A Contour Based Approach without 3D Reconstruction. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 91–96, 1997.

[53] A. Bernstein, R. Lenhardt, J. Hochstrate, and B. Fröhlich. The Haptic SpaceMouse - an Input Device with Force-Feedback through Solenoids. In *Proc. of IEEE VR Workshop New Directions in 3D User Interfaces*, March 2005.

[54] O. Bimber. Spatial Augmented Reality. In *IEEE and ACM ISMAR Tutorials*, page 306, 2004.

[55] O. Bimber, B. Fröhlich, D. Schmalstieg, and M. Encarnação. The Virtual Showcase. *IEEE Computer Graphics and Applications*, 21(6):48–55, November/December 2001.

[56] O. Bimber and R. Raskar. *Spatial Augmented Reality: Merging Real and Virtual Worlds*. A K Peters, July 2005.

[57] W. Birkfellner, K. Huber, F. Watzinger, M. Figl, F. Wanschitz, R. Hanel, D. Rafolt, R. Ewers, and H. Bergmann. Development of the Varioscope AR - A See-through HMD for Computer-Aided Surgery. In *Proc. of IEEE and ACM International Symposium on Augmented Reality (ISAR)*, pages 54–59, October 2000.

[58] U. Bockholt, A. Bisler, M. Becker, W. Müller-Wittig, and G. Voss. Augmented Reality for Enhancement of Endoscopic Interventions. In *Proc. of IEEE Virtual Reality*, pages 97–101, March 2003.

[59] A. Bornik, R. Beichel, B. Reitinger, G. Gotschuli, E. Sorantin, F. Leberl, and M. Sonka. Computer Aided Liver Surgery Planning Based on Augmented Reality Techniques. In *Proc. of Workshop Bildverarbeitung in der Medizin*, pages 249–253, 2003.

[60] A. Bornik, R. Beichel, B. Reitinger, E. Sorantin, G. Werkgartner, F. Leberl, and M. Sonka. EG2003 Medical Prize Competition: Augmented Reality based Liver Surgery Planning. *Computer Graphics Forum*, 22(4):795–796, December 2003.

[61] J. Bortz. *Statistik für Human- und Sozialwissenschaftler*. Springer Verlag, 5th edition, 2005.

[62] BrainLAB AG. BrainLAB Global Site. http://www.brainlab.com, 2006.

[63] BrainLAB AG. Neurosurgery Solutions Brochure, 2006.

[64] D. Breen, R. Whitaker, E. Rose, and M. Tuceryan. Interactive Occlusion and Automatic Object Placement for Augmented Reality. *Computer Graphics Forum*, 15(3):11–22, 1996.

[65] H. Bülthoff, D. Cunningham, B. Adelstein, N. Magnenat-Thalmann, K. Mania, N. Mourkoussis, T. Troscianko, and J. Swan II. Human-Centred Fidelity Metrics for Virtual Environment Simulations. In *IEEE VR Tutorials*, 2005.

[66] E. Burns, S. Razzaque, A. Panter, M. Whitton, M. McCallus, and F. Brooks Jr. The Hand is Slower than the Eye: A Quantitative Exploration of Visual Dominance over Proprioception. In *Proc. of IEEE Virtual Reality*, pages 3–10, 2005.

[67] M. Burns, J. Klawe, S. Rusinkiewicz, A. Finkelstein, and D. DeCarlo. Line Drawings from Volume Data. In *Proc. of ACM SIGGRAPH*, pages 512–518, 2005.

[68] J. Caarls, P. Jonker, and S. Persa. Sensor Fusion for Augmented Reality. In *Proc. of European Symposium on Ambient Intelligence (EUSAI)*, pages 160–176, November 2003.

[69] J. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, November 1986.

[70] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, University of Utah, 1974.

[71] Chicago Institute of Neurosurgery and Neuroresearch. Minimally Invasive Surgery (MIS). http://www.cinn.org/treattech/mis.html, 2006.

[72] D. Cosker, D. Marshall, P. Rosin, S. Paddock, and S. Rushton. Toward Perceptually Realistic Talking Heads: Models, Methods, and McGurk. *ACM Transactions on Applied Perception*, 2(3):270–285, 2005.

[73] C. Cruz-Neira, D. Sandin, and T. DeFanti. Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. In *Proc. of ACM SIGGRAPH*, pages 135–142, 1993.

[74] D. Cunningham, M. Kleiner, C. Wallraven, and H. Bülthoff. Manipulating Video Sequences to Determine the Components of Conversational Facial Expressions. *ACM Transactions on Applied Perception*, 2(3):251–269, 2005.

[75] P. Debevec. Rendering Synthetic Objects Into Real Scenes: Bridging Traditional and Image-Based Graphics With Global Illumination and High Dynamic Range Photography. In *Proc. of ACM SIGGRAPH*, pages 189–198, July 1998.

[76] D. DeCarlo and A. Santella. Stylization and Abstraction of Photographs. In *Proc. of ACM SIGGRAPH*, pages 769–776, July 2002.

[77] P. Decaudin. Cartoon-Looking Rendering of 3D-Scenes. Research Report 2919, INRIA, June 1996.

[78] J. Diepstraten, D. Weiskopf, and T. Ertl. Transparency in Interactive Technical Illustrations. In *Proc. of Eurographics*, pages 317–326, 2002.

[79] J. Diepstraten, D. Weiskopf, and T. Ertl. Interactive Cutaway Illustrations. In *Proc. of Eurographics*, pages 523–532, 2003.

[80] M. Eichler. Modellbasiertes hybrides Tracking für medizinische Augmented Reality. Diplomarbeit, University of Tübingen, November 2005.

[81] M. Eißele, D. Weiskopf, and T. Ertl. The G2-Buffer Framework. In *Proc. of Simulation and Visualization*, pages 287–298, 2004.

[82] Elumens Corporation. Elumens. www.elumens.com, 2006.

[83] J. Ferwerda. Three Varieties of Realism in Computer Graphics. In *Proc. of SPIE Human Vision and Electronic Imaging*, pages 290–297, 2003.

[84] M. Feuerstein, S. Wildhirt, R. Bauernschmitt, and N. Navab. Automatic Patient Registration for Port Placement in Minimally Invasive Endoscopic Surgery. In *Proc. of Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 287–294, October 2005.

[85] M. Figl, W. Birkfellner, C. Ede, J. Hummel, R. Hanel, F. Watzinger, F. Wanschitz, R. Ewers, and H. Bergmann. The Control Unit for a Head Mounted Operating Microscope Used for Augmented Reality Visualization in Computer Aided Sugery. In *Proc. of IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 69–75, September 2002.

[86] M. Figl, W. Birkfellner, J. Hummel, R. Hanel, P. Homolka, F. Watzinger, F. Wanschitz, R. Ewers, and H. Bergmann. Current Status of the Varioscope AR, a Head-Mounted Operating Microscope for Computer-Aided Surgery. In *Proc. of IEEE and ACM International Symposium on Augmented Reality (ISAR)*, pages 20–29, October 2001.

[87] G. Fischer. Electromagnetic Tracker Characterization and Optimal Tool Design (with Applications to ENT Surgery). Master's thesis, John Hopkins University, Baltimore, April 2005.

[88] M. Fischler and R. Bolles. Random Sample Consensus: a Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[89] S. Fisher, M. McGreevy, J. Humphries, and W. Robinett. Virtual Environment Display System. In *Proc. of the Workshop on Interactive 3D graphics*, pages 77–87, October 1986.

[90] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics - Principles and Practice*. Addison-Wesley Publishing Company, 2nd edition, 1997.

[91] B. Freudenberg, M. Masuch, and T. Strothotte. Real-Time Halftoning: A Primitive for Non-Photorealistic Shading. In *Proc. of Eurographics Workshop on Rendering*, pages 227–231, June 2002.

[92] B. Fröhlich, J. Plate, J. Wind, G. Wesche, and M. Göbel. Cubic-Mouse-Based Interaction in Virtual Environments. *IEEE Computer Graphics and Applications*, 20(4):12–15, July 2000.

[93] A. Fuhrmann, G. Hesina, F. Faure, and M. Gervautz. Occlusion in Collaborative Augmented Environments. *Computers & Graphics*, 23(6):809–819, 1999.

[94] J. Gabbard, J. Swan II, D. Hix, R. Schulman, J. Lucas, and D. Gupta. An Empirical User-based Study of Text Drawing Styles and Outdoor Background Textures for Augmented Reality. In *Proc. of IEEE Virtual Reality*, pages 11–18, 2005.

[95] S. Gibson, T. Howard, and R. Hubbold. Flexible Image-Based Photometric Reconstruction using Virtual Light Sources. In *Proc. of Eurographics*, pages 203–214, September 2001.

[96] G. Goebbels. ARSyS-Tricorder - Entwicklung eines Augmented Reality Systems für die intraoperative Navigation in der MKG Chirurgie. In *Jahrestagung der Deutschen Gesellschaft für Computer-und Roboterassistierte Chirurgie e.V. (CURAC)*, November 2003.

[97] R. Gonzalez and R. Woods. *Digital Image Processing*. Prentice-Hall, 2nd edition, 2002.

[98] A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A Non-Photorealistic Lighting Model For Automatic Technical Illustration. In *Proc. of ACM SIGGRAPH*, pages 447–452, 1998.

[99] B. Gooch and A. Gooch. *Non-Photorealistic Rendering*. A K Peters, 2nd edition, 2001.

[100] G. Gordon, M. Billinghurst, M. Bell, J. Woodfill, B. Kowalik, A. Erendi, and J. Tilander. The Use of Dense Stereo Range Data in Augmented Reality. In *Proc. of IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 14–26, September 2002.

[101] R. Grasset, L. Boissieux, J. Gascuel, and D. Schmalstieg. Interactive mediated reality. In *Proc. of Australasian conference on User interface (CRPIT)*, pages 21–29, 2005.

[102] R. Grasset, J. Gascuel, and D. Schmalstieg. Interactive Mediated Reality. In *IEEE and ACM International Symposium on Mixed and Augmented Reality Poster Proceedings*, pages 302–303, October 2003.

[103] M. Haller. Photorealism or/and Non-Photorealism in Augmented Reality. In *Proc. of ACM SIGGRAPH International Conference on Virtual Reality Continuum and its Applications in Industry (VRCAI)*, pages 189–196, June 2004.

[104] M. Haller, S. Drab, W. Hartmann, and J. Zauner. A Real-time Shadow Approach for an Augmented Reality Application using Shadow Volumes. In *Proc. of ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 56–65, October 2003.

[105] M. Haller, F. Landerl, and M. Billinghurst. A Loose and Sketchy Approach in a Mediated Reality Environment. In *Proc. of ACM International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (Graphite)*, pages 371–379, December 2005.

[106] M. Haller and D. Sperl. Real-Time Painterly Rendering for MR Applications. In *Proc. of ACM International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (Graphite)*, pages 30–38, June 2004.

[107] M. Haringer and H. Regenbrecht. A Pragmatic Approach to Augmented Reality Authoring. In *Proc. of IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 137–145, 2002.

[108] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2004.

[109] A. Hertzmann. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. In *Proc. of ACM SIGGRAPH*, pages 453–460, 1998.

[110] A. Hertzmann and K. Perlin. Painterly Rendering for Video and Interaction. In *Proc. of ACM International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 7–12, June 2000.

[111] A. Hertzmann and D. Zorin. Illustrating Smooth Surfaces. In *Proc. of ACM SIGGRAPH*, pages 517–526, 2000.

[112] S. Heymann, A. Smolic, K. Müller, and B. Fröhlich. Illumination Reconstruction from Real-Time Video For Interactive Augmented Reality. In *Proc. of International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, April 2005.

[113] S. Heymann, A. Smolic, K. Müller, and B. Fröhlich. Real-Time Video Capture for Illumination Reconstruction in Augmented Reality Applications. In *Proc. of ACM Symposium on Interactive 3D Graphics*, April 2005.

[114] J. Hoffmann, D. Troitzsch, M. Schneider, F. Reinauer, and D. Bartz. Evaluation der 3-D-Präzision eines bilddatengestützten chirurgischen Navigationssystems. In *Proc. of Workshop Bildverarbeitung in der Medizin*, pages 289–292, 2003.

[115] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual Voyage: Interactive Navigation in the Human Colon. In *Proc. of ACM SIGGRAPH*, pages 27–34, August 1997.

[116] Intel Corporation. *Open Source Computer Vision Library Reference Manual*, 2001.

[117] V. Interrante, H. Fuchs, and S. Pizer. Conveying the 3D Shape of Smoothly Curving Transparent Surfaces via Texture. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):98–117, March 1997.

[118] M. Juan, M. Alcañiz, C. Monserrat, C. Botella, R. Baños, and B. Guerrero. Using Augmented Reality to Treat Phobias. *IEEE Computer Graphics and Applications*, 25(6):31–37, November/December 2005.

[119] S. Julier, A. Davison, and A. Fitzgibbon. Advanced Visual Tracking. In *IEEE and ACM ISMAR Tutorials*, page 305, 2004.

[120] M. Kanbara, T. Okuma, H. Takemura, and N. Yokoya. A Stereoscopic Video See-through Augmented Reality System Based on Real-time Vision-based Registration. In *Proc. of IEEE Virtual Reality*, pages 255–262, March 2000.

[121] M. Kanbara and N. Yokoya. Geometric and Photometric Registration for Real-Time Augmented Reality. In *IEEE and ACM International Symposium on Mixed and Augmented Reality Poster Proceedings*, pages 279–280, September 2002.

[122] M. Kaplan, B. Gooch, and E. Cohen. Interactive Artistic Rendering. In *Proc. of ACM International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 67–74, 2000.

[123] H. Kato and M. Billinghurst. Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System. In *Proc. of IEEE and ACM International Workshop on Augmented Reality (IWAR)*, pages 85–94, October 1999.

[124] J. Kessenich, D. Baldwin, and R. Rost. The OpenGL® Shading Language (v1.10). http://www.opengl.org/documentation/oglsl.html, 2004.

[125] S. Kim, H. Hagh-Shenas, and V. Interrante. Conveying Shape with Texture: experimental investigations of texture's effects on shape categorization judgments. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):471–483, 2004.

[126] A. Klein, W. Li, M. Kazhdan, W. Correa, A. Finkelstein, and T. Funkhouser. Non-Photorealistic Virtual Environments. In *Proc. of ACM SIGGRAPH*, pages 527–534, July 2000.

[127] G. Klinker. Einführung in die Erweiterte Realität - 3. OpenGL. campar.in.tum.de, 2003.

[128] M. Köbele. Augmented-Reality Based Classification for Volume Rendering in Medical Visualisation. Diplomarbeit, University of Tübingen, April 2005.

[129] W. Krüger, C. Bohn, B. Fröhlich, H. Schüth, W. Strauss, and G. Wesche. The Responsive Workbench: A Virtual Work Environment. *IEEE Computer*, 28(7):42–48, 1995.

[130] J. Lander. Shades of Disney: Opaquing a 3D World. *Game Developer Magazine*, 7(3):15–20, March 2000.

[131] J. Lander. Under the Shade of the Rendering Tree. *Game Developer Magazine*, 7(2):17–21, February 2000.

[132] F. Ledermann and D. Schmalstieg. APRIL: A High-level Framework for Creating Augmented Reality Presentations. In *Proc. of IEEE Virtual Reality*, pages 187–194, 2005.

[133] V. Lepetit and M.-O. Berger. A Semi-Automatic Method for Resolving Occlusion in Augmented Reality. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2225–2230, June 2000.

[134] M. Levoy. The Stanford volume data archive. http://www-graphics.stanford.edu/data/voldata/, 2004.

[135] W. Lin, K. Sengupta, P. Kumar, and R. Sharma. Occlusion Handling in Augmented Reality Using Background Foreground Segmentation. *Presence: Teleoperators and Virtual Environments*, 14(3):264–277, June 2005.

[136] P. Longhurst, P. Ledda, and A. Chalmers. Psychophysically Based Artistic Techniques for Increased Perceived Realism of Virtual Environments. In *Proc. of the ACM International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (Afrigraph)*, pages 123–132, 2003.

[137] W. Lorensen and H. Cline. Marching Cubes: A High Resolution 3d Surface Construction Algorithm. In *Proc. of ACM SIGGRAPH*, pages 163–169, July 1987.

[138] A. Lu, C. Morris, D. Ebert, P. Rheingans, and C. Hansen. Non-Photorealistic Volume Rendering Using Stippling Techniques. In *Proc. of IEEE Visualization*, pages 211–218, 2002.

[139] E. Lum and K. Ma. Non-Photorealistic Rendering Using Watercolor Inspired Textures and Illumination. In *Proc. of Pacific Conference on Computer Graphics and Applications*, pages 322–331, 2001.

[140] K. Mania, B. Adelstein, S. Ellis, and M. Hill. Perceptual Sensitivity to Head Tracking Latency in Virtual Environments with Varying Degrees of Scene Complexity. In *Proc. of the Symposium on Applied Perception in Graphics and Visualization (APGV)*, pages 39–47, 2004.

[141] K. Mania and A. Robinson. An Experimental Exploration of the Relationship between Subjective Impressions of Illumination and Physical Fidelity. *Computers & Graphics*, 29:49–56, 2005.

[142] S. Mann. 'Mediated Reality'. Technical Report 260, M.I.T. Media Lab Perceptual Computing Section, Cambridge, Massachusetts, http://wearcam.org/mr.htm, 1994.

[143] S. Mann. Wearable Computing: A First Step Toward Personal Imaging. *IEEE Computer*, 30(2):25–32, February 1997.

[144] A. McNamara. Exploring Perceptual Equivalence between Real and Simulated Imagery. In *Proc. of the Symposium on Applied Perception in Graphics and Visualization (APGV)*, pages 123–128, 2005.

[145] B. Meier. Painterly Rendering for Animation. In *Proc. of ACM SIGGRAPH*, pages 477–484, 1996.

[146] M. Mekić. Ein medizinisches Planungswerkzeug für die Orbita Rekonstruktion. Diplomarbeit, Universität Tübingen, 2003.

[147] J. Mitchell, C. Brennan, and D. Card. Real-Time Image Space Outlining for Non-Photorealistic Rendering. In *ACM SIGGRAPH Sketches*, page 239, 2002.

[148] J. Murray and W. Van Ryper. Wavefront OBJ File Format Summary. http://www.fileformat.info/format/wavefrontobj/, 2005.

[149] N. Nachlas. Image - Guided Surgery. eMedicine.com, Inc, http://www.emedicine.com/ent/topic396.htm, January 2003.

[150] N. Navab, A. Bani-Hashemi, and M. Mitschke. Merging Visible and Invisible: Two Camera-Augmented Mobile C-arm (CAMC) Applications. In *Proc. of IEEE and ACM International Workshop on Augmented Reality (IWAR)*, pages 134–141, October 1999.

[151] M. Neff and K. Diepold. Design and Implementation of an Interface Facilitating Data Exchange between an IGS System and External Image Processing Software. Diplomarbeit, TU München, May 2003.

[152] M. Nienhaus and J. Döllner. Edge-Enhancement - An Algorithm for Real-Time Non-Photorealistic Rendering. In *Proc. of WSCG*, pages 346–353, 2003.

[153] M. Nienhaus and J. Döllner. Blueprints - Illustrating Architecture and Technical Parts using Hardware-Accelerated Non-Photorealistic Rendering. In *Proc. of Graphics Interface*, pages 49–56, 2004.

[154] F. Nooruddin and G. Turk. Interior/Exterior Classification of Polygonal Models. In *Proc. of IEEE Visualization*, pages 415–422, 2000.

[155] B. Okumura, M. Kanbara, and N. Yokoya. Image Composition Based on Blur Estimation from Captured Image for Augmented Reality. In *IEEE and ACM ISMAR Demo Session*, 2005.

[156] B. Okumura, M. Kanbara, and N. Yokoya. Image Composition Based on Blur Estimation from Captured Image for Augmented Reality. In *Proc. of IEEE Virtual Reality*, pages 128–134, 2006.

[157] B. Olbrich, J. Traub, S. Wiesner, A. Wiechert, H. Feußner, and N. Navab. Respiratory Motion Analysis: Towards Gated Augmentation of the Liver. In *Proc. of Computer Assisted Radiology and Surgery*, pages 248–253, June 2005.

[158] H. Oosterwijk and P. Gihring. *DICOM Basics*. OTech, Inc., 2nd edition, 2002.

[159] R. Pausch, D. Proffitt, and G. Williams. Quantifying Immersion in Virtual Reality. In *Proc. of ACM SIGGRAPH*, pages 13–18, August 1997.

[160] H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. Avila, R. Machiraju, and J. Lee. The Transfer Function Bake-off. *IEEE Computer Graphics and Applications*, 21(3):16–22, 2001.

[161] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface Elements as Rendering Primitives. In *Proc. of ACM SIGGRAPH*, pages 335–342, July 2000.

[162] W. Piekarski, B. Avery, B. Thomas, and P. Malbezin. Hybrid Indoor and Outdoor Tracking for Mobile 3D Mixed Reality. In *IEEE and ACM International Symposium on Mixed and Augmented Reality Poster Proceedings*, pages 266–267, October 2003.

[163] W. Pratt. *Digital Image Processing*. John Wiley & Sons, 3rd edition, 2001.

[164] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-Time Hatching. In *Proc. of ACM SIGGRAPH*, pages 579–584, 2001.

[165] P. Rademacher, J. Lengyel, E. Cutrell, and T. Whitted. Measuring the Perception of Visual Realism in Images. In *Proc. of Eurographics Workshop on Rendering*, pages 235–248, 2001.

[166] H. Regenbrecht, G. Baratoff, and W. Wilke. Augmented Reality Projects in Automotive and Aerospace Industry. *IEEE Computer Graphics and Applications*, 25(6):48–56, November/December 2005.

[167] H. Regenbrecht and T. Schubert. Measuring Presence in Augmented Reality Environments: Design and a First Test of a Questionnaire. In *Proc. of International Workshop on Presence*, October 2002.

[168] P. Rheingans and D. Ebert. Volume Illustration: Nonphotorealistic Rendering of Volume Models. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):253–264, July-September 2001.

[169] R. Rost. *OpenGL Shading Language*. Addison-Wesley Publishing Company, 2004.

[170] T. Saito and T. Takahashi. Comprehensible Rendering of 3-D Shapes. In *Proc. of ACM SIGGRAPH*, pages 197–206, 1990.

[171] F. Sauer, A. Khamene, B. Bascle, L. Schimmang, F. Wenzel, and S. Vogt. Augmented Reality Visualization of Ultrasound Images: System Description, Calibration, and Features. In *Proc. of IEEE and ACM International Symposium on Augmented Reality (ISAR)*, pages 30–44, October 2001.

[172] C. Schaller. IsoMax - Prototyp eines Planungswerkzeugs für die MKG-Chirurgie auf Basis polygonaler Knochendarstellung. Studienarbeit, Universität Tübingen, 2005.

[173] M. Scheuering, C. Rezk-Salama, H. Barfuss, A. Schneider, and G. Greiner. Augmented Reality Based On Fast Deformable 2D-3D Registration For Image Guided Surgery. In *Proc. of SPIE Medical Imaging*, pages 436–445, 2002.

[174] M. Scheuering, A. Schenk, A. Schneider, B. Preim, and G. Greiner. Intraoperative Augmented Reality for Minimally Invasive Liver Interventions. In *Proc. of SPIE Medical Imaging*, pages 407–417, 2003.

[175] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach To 3D Graphics*. Prentice Hall, 2nd edition, December 1997.

[176] W. Schroeder, J. Zarge, and W. Lorensen. Decimation of Triangle Meshes. In *Proc. of ACM SIGGRAPH*, pages 65–70, July 1992.

[177] T. Schubert and H. Regenbrecht. Wer hat Angst vor virtueller Realität? Phobie, Therapie und Präsenz in virtuellen Welten. In *Proc. of Virtuelle Realitäten*, pages 255–274, 2002.

[178] B. Schwald and H. Seibert. Registration Tasks for a Hybrid Tracking System for Medical Augmented Reality. In *Proc. of WSCG*, pages 411–418, February 2004.

[179] B. Schwald, H. Seibert, and T. Weller. A Flexible Tracking Concept Applied to Medical Scenarios Using an AR Window. In *IEEE and ACM International Symposium on Mixed and Augmented Reality Poster Proceedings*, pages 261–262, September 2002.

[180] A. Secord, W. Heidrich, and L. Streit. Fast Primitive Distribution for Illustration. In *Proc. of Eurographics Workshop on Rendering*, pages 215–226, 2002.

[181] D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide*. Addison-Wesley Publishing Company, 4th edition, 2003.

[182] D. Sperl. Realtime Painterly Rendering for Animation. Technical report, Hagenberg College of Information Technology, Austria, 2003.

[183] R. Splechtna, A. Fuhrmann, and R. Wegenkittl. ARAS - Augmented Reality Aided Surgery System Description. Technical report, VRVis Research Center for Virtual Reality and Visualization, 2002.

[184] D. Staneker, D. Bartz, and W. Straßer. Occlusion Culling in OpenSG PLUS. *IEEE Computer Graphics and Applications*, 28(1):87–92, 2004.

[185] A. State, G. Hirota, D. Chen, W. Garrett, and M. Livingston. Superior Augmented-Reality Registration by Integrating Landmark Tracking and Magnetic Tracking. In *Proc. of ACM SIGGRAPH*, pages 429–438, August 1996.

[186] A. State, M. Livingston, G. Hirota, W. Garrett, M. Whitton, H. Fuchs, and E. Pisano. Technologies for Augmented-Reality Systems: Realizing Ultrasound-Guided Needle Biopsies. In *Proc. of ACM SIGGRAPH*, pages 439–446, August 1996.

[187] StatSoft Inc. ANOVA/MANOVA. http://www.statsoft.com/textbook/stanman.html, 2003.

[188] W. Straßer. *Schnelle Kurven- und Flächendarstellung auf graphischen Sichtgeräten*. Dissertation, Technical University of Berlin, 1974.

[189] T. Strothotte and S. Schlechtweg. *Non-Photorealistic Computer Graphics - Modelling, Rendering, and Animation*. Morgan Kaufmann Publishers, 2002.

[190] N. Sugano, H. Kato, and K. Tachibana. The Effects of Shadow Representation of Virtual Objects in Augmented Reality. In *Proc. of IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 76–83, October 2003.

[191] I. Sutherland. A Head-Mounted Three-Dimensional Display. In *Proc. of the AFIPS Fall Joint Computer Conference*, volume 33, pages 757–764, 1968.

[192] F. Tang, C. Aimone, J. Fung, A. Marjan, and S. Mann. Seeing Eye to Eye: A Shared Mediated Reality Using EyeTap Devices and the VideoOrbits Gyroscopic Head Tracker. In *IEEE and ACM International Symposium on Mixed and Augmented Reality Poster Proceedings*, pages 267–268, September 2002.

[193] B. Thomas, B. Close, J. Donoghue, J. Squires, P. de Bondi, M. Morris, and W. Piekarski. ARQuake: An Outdoor/Indoor Augmented Reality First Person Application. In *Proc. of IEEE International Symposium on Wearable Computers*, pages 139–146, 2000.

[194] C. Tomasi and R. Manduchi. Bilateral Filtering for Gray and Color Images. In *Proc. of IEEE International Conference on Computer Vision (ICCV)*, pages 839–846, 1998.

[195] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall PTR, 1998.

[196] University of North Carolina at Chapel Hill, Department of Computer Science. UNC Ultrasound / Medical Augmented Reality Research. http://www.cs.unc.edu/Research/us/, 2000.

[197] J. Vince. *Virtual Reality Systems*. Addison-Wesley, 1995.

[198] I. Viola, A. Kanitsar, and E. Gröller. Importance-Driven Volume Rendering. In *Proc. of IEEE Visualization*, pages 139–145, 2004.

[199] I. Viola, A. Kanitsar, and M. Gröller. Hardware-Based Nonlinear Filtering and Segmentation using High-Level Shading Languages. In *Proc. of IEEE Visualization*, pages 309–316, October 2003.

[200] V. Vlahakis, J. Karigiannis, L. Almeida, D. Stricker, T. Gleue, I. Christou, R. Carlucci, and N. Ioannidis. ARCHEOGUIDE: First Results of an Augmented Reality, Mobile Computing System in Cultural Heritage Sites. In *Proc. of Conference on Virtual Reality, Archeology, and Cultural Heritage (VAST)*, pages 131–140, 2001.

[201] C. Wallraven, M. Breidt, D. Cunningham, and H. Bülthoff. Psychophysical Evaluation of Animated Facial Expressions. In *Proc. of the Symposium on Applied Perception in Graphics and Visualization (APGV)*, pages 17–24, 2005.

[202] J. Wang, Y. Xu, H. Shum, and M. Cohen. Video Tooning. In *Proc. of ACM SIGGRAPH*, pages 574–583, August 2004.

[203] M. Webb, E. Praun, A. Finkelstein, and H. Hoppe. Fine Tone Control in Hardware Hatching. In *Proc. of ACM International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 53–58, June 2002.

[204] S. Weber. *Ein navigierter Bildbetrachter für medizinische Bilddaten*. Dissertation, Humboldt-Universität zu Berlin, January 2003.

[205] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991.

[206] B. Witmer and M. Singer. Measuring Presence in Virtual Environments: A Presence Questionnaire. *Presence: Teleoperators and Virtual Environments*, 7(3):225–240, June 1998.

[207] M. Wloka and B. Anderson. Resolving Occlusion in Augmented Reality. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 5–12, 1995.

[208] Wolfram Research. MathWorld: Voronoi Diagram. http://mathworld.wolfram.com/, 2005.

[209] D. Wormell and E. Foxlin. Advancements in 3D Interactive Devices for Virtual Environments. In *Proc. of Eurographics Symposium on Virtual Environments*, pages 47–55, May 2003.

[210] H. Xu and B. Chen. Stylized Rendering of 3D Scanned Real World Environments. In *Proc. of ACM International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 25–34, 2004.

[211] S. You, U. Neumann, and R. Azuma. Hybrid Inertial and Vision Tracking for Augmented Reality Registration. In *Proc. of IEEE Virtual Reality*, pages 260–267, March 1999.

[212] X. Yuan and B. Chen. Illustrating Surfaces in Volume. In *Data Visualization (Proc. of Eurographics/IEEE VGTC Symposium on Visualization)*, pages 9–16, 2004.

APPENDIX A

# The Virtue Framework

## A.1 Overview of the Virtue Repository

The research efforts described in this thesis were implemented based on a specialized new software framework, which was developed and extended in the context of this thesis. This software framework consists of a set of libraries which provide support for many of the basic tasks required for the research topics presented here. These basic tasks include general utility functionality like handling debug output and time measurements, file input/output, three-dimensional mathematics through linear algebra classes, a specialized scene graph library for customized rendering, and network communication with the image guided surgery device, as well as a framework for augmented reality applications. The software framework is called *Virtue* after the grant from the Deutsche Forschungsgemeinschaft (German Research Foundation) which funded this thesis. The entire *Virtue* framework exists as a single CVS (Concurrent Versions System) repository of the same name. As one part of the framework, the augmented reality software foundation *ARGUS* is contained in a subdirectory. The complete software foundation as well as all developments based on it were written in the C++ programming language using object oriented design methodology.

The following directories are contained in the *Virtue* repository. Among them, the most important directories are `core/`, `vvl/`, and `ar/`. Each of these corresponds to one library file, and each of these has its own set of dependencies on third-party software.

| | |
|---:|---|
| `apps/` | Applications based on the *Virtue* framework. |
| `core/` | *Virtue* core library sources, headers and test programs. |
| `doc/` | Documentation for the *Virtue* project. |
| `lib/` | Output directory for the generated library files. |
| `VirtueSolution/` | Visual Studio .NET project files (only for the Windows OS). |
| `vvl/` | *Virtue* VectorVision Link library sources and test programs. |
| `ar/` | *Virtue* augmented reality library sources and test programs, i.e., the *ARGUS* framework. |
| `Helios/` | *Helios* sources, headers and test programs. It includes the ’fire’ subdirectory containing resources for isosurface handling. |

In the remainder of this appendix, Section A.2 will demonstrate the use of the *Virtue* scene graph with a simple example, and Section A.3 will show how to use the *ARGUS* framework.

## A.2 Virtue Scene Graph Example

The *Virtue* scene graph is a C++ scene graph library contained in the `core` module of the repository. While it is not particularly optimized for the fast rendering of very large or very complex models and does not have its own native file format, its uncomplicated structure and easy extensibility make it ideal for research purposes.

Each node in the *Virtue* scene graph is derived from the class `VirSceneNode`, a fully abstract class which only consists of the interface for the required `traverse()` method. Each scene graph node implements this method, either by recursively traversing child nodes or generating the appropriate OpenGL calls. The main structuring element of a *Virtue* scene graph is the `VirGroupNode` class. `VirGroupNode` is the superclass of all group nodes used in the scene graph. Each group node can hold pointers to an arbitrary number of child nodes. Whenever a group node is traversed, it recursively traverses its child nodes. The management of child nodes is done with an interface which is equivalent to many other scene graph libraries, providing methods like `addChild()`, `getChild()`, and `removeChild()`.

**Algorithm 8** Declaration of instance variables for the scene graph example.

```
//----- Scene graph objects

// Transformations

VirSeparator *root;

VirTranslationGroup *cartoonTransG;
VirRotationGroup *cartoonRotG;
VirScaleGroup *cartoonScaleG;
VirTranslationGroup *cartoonOriginTransG;

VirSwitchNode *renderModeSwitchG;

// Normal rendering mode

VirGroupNode *normalRenderG;
VirDirLightNode *normalRenderLight;
VirMaterialNode *normalRenderMat;
VirEnableTexture2DNode *normalRenderTexEnable;
VirTexture2DNode *normalRenderTex;
VirVtkPolyDataNode *normalRenderPolyD;
```

In the following, a simple example demonstrating the use of the scene graph will be discussed. Algorithm 8 shows the declaration of instance variables in the example application class. These instance variables are pointers to scene graph nodes, to which newly created node objects are later assigned. The root of the scene graph is a `VirSeparator` node named `root`. The `VirSeparator` class is a group node which encapsulates the current OpenGL state. Subsequently, a number of different other types of group nodes are declared. These group node classes, e.g., `VirTranslationGroup` and `VirScaleGroup`, are the transformation groups in the *Virtue* scene graph. They apply some kind of geometric transformation to their child nodes and therefore constitute the core of the hierarchical modelling approach realized in the scene graph. At the end of the code shown in Algorithm 8, a number of leaf nodes are declared. These scene graph leaf nodes generate OpenGL primitives (e.g., `VirVtkPolyDataNode`) or manipulate the OpenGL rendering parameters (e.g., `VirMaterialNode`). Within one group node, child nodes are traversed in sequential order, i.e., in the order in which they were added to the group ("from left to right").

Algorithm 9 shows the construction of the example scene graph. Each of the node pointers is initialized with a newly constructed object of the respective type. The scene graph hierarchy is then constructed with calls to `addChild()`, which adds a node pointer as child to a group node. In the second part of the code, the leaf nodes are constructed and added to the group node at the bottom of the hierarchy (`normalRenderG`). For some of the nodes, attributes determining their behavior are set, e.g., using `getDirection().set()` for a light node. The example scene graph is shown in Figure A.1.

**Algorithm 9** Construction of the example scene graph.

```
//----- Scene graph setup

root = new VirSeparator();

// Transformations

cartoonTransG = new VirTranslationGroup();
root->addChild(cartoonTransG);

cartoonRotG = new VirRotationGroup();
cartoonTransG->addChild(cartoonRotG);

cartoonScaleG = new VirScaleGroup();
cartoonRotG->addChild(cartoonScaleG);

cartoonOriginTransG = new VirTranslationGroup();
cartoonScaleG->addChild(cartoonOriginTransG);

renderModeSwitchG = new VirSwitchNode();
cartoonOriginTransG->addChild(renderModeSwitchG);
renderModeSwitchG->setChildToTraverse(0);

// Scene graph leaf nodes

normalRenderG = new VirGroupNode();
renderModeSwitchG->addChild(normalRenderG);

normalRenderLight = new VirDirLightNode();
normalRenderLight->getDirection().set(0.0, 0.0, -1.0);
normalRenderLight->setUseIdentityTransformation(true);
normalRenderG->addChild(normalRenderLight);

normalRenderMat = new VirMaterialNode();
normalRenderG->addChild(normalRenderMat);

normalRenderTexEnable = new VirEnableTexture2DNode(false);
normalRenderG->addChild(normalRenderTexEnable);

normalRenderTex = new VirTexture2DNode();
normalRenderG->addChild(normalRenderTex);

normalRenderPolyD = new VirVtkPolyDataNode();
normalRenderG->addChild(normalRenderPolyD);
```

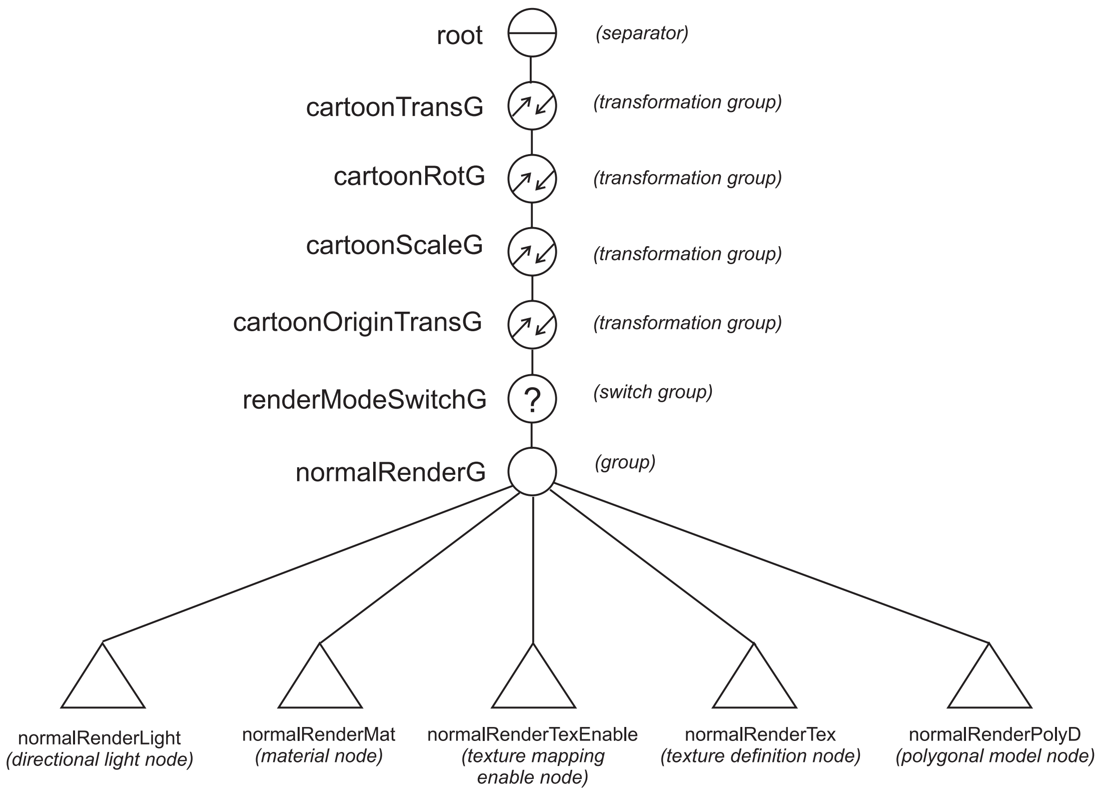


Figure A.1: The example scene graph.

In the rendering method of the application, the scene can easily be rendered by calling the traverse method of the root node, i.e., `root->traverse()`. This way, the complete hierarchy is iteratively traversed, and the entire scene is displayed. In order to deallocate the scene graph when the application object is deleted, only the root node has to be deallocated with `delete root`. It is the default behavior of group nodes to recursively delete their children, leading to an automatic deallocation of the entire scene graph. This also entails the automatic release of all OpenGL resources allocated by the scene graph (e.g., display lists or texture objects).

## A.3 ARGUS Application Example

The *ARGUS* framework is a foundation for augmented reality applications, which served as the basis for all AR developments described in this thesis. *ARGUS* is based on the ARToolKit library, which provides the functionality for video acquisition, camera calibration, and video-based marker tracking. However, unlike the original ARToolKit software, *ARGUS* provides an advanced graphical user interface (GUI) using the Qt toolkit. In order to make the use of Qt in connection with the ARToolKit software possible, a modified version of the ARToolKit library was created. The source code of this modified library also resides in the *Virtue* repository.

Two main classes constitute the core of the *ARGUS* framework. The `ArgMainWindow` class is a Qt main window containing the central GUI for any *ARGUS* application. It holds some of the data required for running augmented reality applications and defines the drop-down menus, menu items, and several dialogs which make up the user interface of the AR application. Algorithm 10 demonstrates the construction of a simple *ARGUS* main window. The application code simply has to derive a window class of its own from `ArgMainWindow`. The main window class then typically holds a pointer to an object derived from the second central *ARGUS* class, `ArgGLWidget`. This pointer is initialized in the constructor with a newly allocated object. As one peculiarity of the *ARGUS* framework, the `ArgGLWidget`-derived object must be made known to the main window using the method `setArgGLWidget`, preferably during the initialization process. This way, the correct connections between the main window and its OpenGL widget are created.

Algorithm 11 shows an example for the typical use of the `ArgGLWidget` class. The *ARGUS* application class derives a class of its own from `ArgGLWidget`. The superclass provides all the necessary functionality for running the actual augmented reality application, e.g., video acquisition and marker tracking, as well as some simple standard graphical displays. The derived class only has to implement some abstract methods, in particular `initializeAR()` and `renderAR()`. These replace the normal OpenGL callbacks found in conventional windowing frameworks. In `initializeAR()`, all the required OpenGL initialization code should be executed. `renderAR()` is the central method of the *ARGUS* OpenGL widget class and is supposed to contain the application-specific rendering code. In the example code in Algorithm 11, an example scene graph is constructed in `initSceneGraph()` (details are left out of the code) and then rendered with a call to `root->traverse()` in `renderAR()`.

In order to generate a basic *ARGUS* application, it is sufficient to derive application-specific classes from `ArgMainWindow` and `ArgGLWidget` and create the corresponding objects. The main window contains the required user interface for basic augmented reality tasks, but also for infrared tracking and calibration using the image guided surgery device. Moreover, some utility functionality like benchmarking options and frame grabbing are provided. The *ARGUS* application can extend this user interface in order to provide access to its own functionality.

**Algorithm 10** Example code for a simple *ARGUS* main window.

Main window header file (`ArtMainWindow.h`):

```
#include "ArgMainWindow.h"

class ArtGLWidget;

class ArtMainWindow :  public ArgMainWindow {
    private:
        QWidget *arParentWidget;
        ArtGLWidget *artGLWidget;

        void makeUI();

    protected:
    public:
        ArtMainWindow(QApplication *app = 0, QWidget *parent = 0,
                      const char *name = 0);
        ~ArtMainWindow();
};
```

Main window implementation file (`ArtMainWindow.cpp`):

```
#include "ArtGLWidget.h"

#include "ArtMainWindow.h"

void ArtMainWindow::makeUI()
{
    arParentWidget = new QWidget(this);
    setCentralWidget(arParentWidget);

    artGLWidget = new ArtGLWidget(arParentWidget, getTheApp());
    setArgGLWidget(artGLWidget);
}

ArtMainWindow::ArtMainWindow(QApplication *app, QWidget *parent,
                             const char *name)
    :  ArgMainWindow(app, parent, name)
{
    setCaption("ARGUS Test");
    makeUI();
}

ArtMainWindow::~ArtMainWindow()
{ }
```

**Algorithm 11** Example code for a simple *ARGUS* OpenGL widget.

OpenGL widget header file (`ArtGLWidget.h`):

```
#include "ArgGLWidget.h"

class ArtGLWidget :  public ArgGLWidget {
    private:
        void initSceneGraph();

    protected:
        virtual void initializeAR();
        virtual void renderAR();

    public:
        ArtGLWidget(QWidget *parent = 0, QApplication *_app = 0,
                    const char *name = 0);
        virtual ~ArtGLWidget();
};
```

OpenGL widget implementation file (`ArtGLWidget.cpp`):

```
#include "ArtGLWidget.h"

void ArtGLWidget::initSceneGraph()
{
    // Build scene graph
}

void ArtGLWidget::initializeAR()
{
    initSceneGraph();
}

void ArtGLWidget::renderAR()
{
    root->traverse();
}

ArtGLWidget::ArtGLWidget(QWidget *parent, QApplication *_app,
                         const char *name)
    :  ArgGLWidget(parent, _app, name)
{ }

ArtGLWidget::~ArtGLWidget()
{ }
```

APPENDIX B

# Coding Conventions

## B.1 Introduction

In this appendix, the coding conventions applied for the software framework and research projects described in this thesis are discussed. This coding styleguide mainly covers rules for C++ programming, but also some general project management aspects. The complete original styleguide is reproduced in Section B.2. These coding conventions were also used for student and diploma theses based on the *Virtue* framework, as well as some other related projects like *Helios/Fire* (by Ángel del Río) and *Volv* (by Matthias Pfeifle).

## B.2 The Original Coding Styleguide of the Virtue Project

### Programming Language, Features, Portability

All parts of the project are to be written in pure C++, using object oriented methodology wherever possible. The use of global variables, functions and constants should be strictly limited to cases where they are absolutely unavoidable. An example of this are the (rare) callback mechanisms of some APIs (like GLUT's display callback or Open Inventor's SoCallback node).

All data and functionality beyond what is absolutely required for the main program is to be put into C++ classes. Classical C constructs like structs and #define constants and macros should be avoided in favor of their typesafe and access-limited C++ counterparts.

As much code as possible should be independent from the platform used (OS and/or computer system). If any part of the code is specific to a certain platform, it should be encapsuled, preferably in a class of its own. Conditional compilation for this purpose should be used as little as possible.

## Object Oriented Programming

Whenever data types and classes share a subset of their data structure or functionality, it should be put into a common base class.

Within each class, **all member variables are to be declared as private variables** to ensure data encapsulation. Access to member data should be only possible through special access methods. In order to provide a uniform way of accessing private variables, a simple naming convention is to be followed for these access methods (see Section “Methods”).

The **extremely rare** exception to the private-variables-only rule are classes which consist of public variables only. Such classes have a similar role as data structures in pure C programs. These data-only classes provide a fast way of manipulating data structures, but can only be used when no data consistency must be ensured. An example of such a data-only class is a simple 3-d vector class:

```
class Vector3d {
    public:
        double x, y, z;

        Vector3d() { }
        ~Vector3d() { }
};
```

## Base APIs and Libraries to be used

The following libraries and APIs should be used preferredly whenever the respective functionality is required:

- C++ standard library and IOStreams library (advanced data structures, algorithms, standard I/O)
- Qt (GUI, OpenGL rendering context, certain OS-independent services, XML services)
- OpenGL (all types of hardware-accelerated rendering)
- GLEW, the OpenGL Extension Wrangler Library (access to OpenGL extensions, and in particular the OpenGL Shading Language)
- VTK for volume input/output, direct volume rendering, basic image processing and isosurface extraction
- Classes from the `Virtue` repository for the OpenGL-based `Virtue` scene graph, debug output classes, benchmarking functionality, AR base functionality etc. for software written in the context of `Virtue` and `ARGUS` projects

The use of system-specific libraries and APIs (e.g. the Win32 API or Linux-specific functions) is to be avoided.

## Editor Settings

The C++ source code has to be saved **without any tab characters**. Indentations must always be converted to regular spaces to ensure compatibility with as many editors as possible.

As explained below, indentations should consist of **four space characters** each.

## Directory and File Structure

Each project should be given a directory of its own. If the project is important or consists of many files, there should be an `include` and a `src` directory for the header and the code files respectively.

```
MyProject
+-include
+-src
```

Compiler and IDE specific files and directory structures (e.g. Visual Studio Workspaces) should be located seperately to keep the project sourcecode itself as pure as possible.

```
MyProject
+-include
+-src
+-VSWorkpace
```

Simple/small projects (like test programs) can have a flat directory structure, which can be located within the main project directory.

```
MyProject
+-include
+-src
+-tests
  +-IOTest (sources and headers)
  +-DebugTest (sources and headers)
+-VSWorkspace
```

Big projects can also be split into several subdirectories, each containing both an `include` and a `src` directory.

C++ source files must have the file extension `.cpp`, header files the extension `.h`. Each pair of source and header file should normally correspond to a single C++ class. **The filenames should be exactly the same as the name of the contained class (including case-sensitivity)**. Template classes are the exception to this rule. Template classes are to be written exclusively in a header file in order to avoid compilation problems.

Qt designer files (`*.ui` and `ui.h`) and qmake project files (`*.pro`) should be placed in the directory of the respective source code.

## Naming Conventions

All names and identifiers used in a project should be chosen to be as understandable and self-explanatory as possible. Long and readable identifiers should be used rather than short ones. All names must be in **English** (including variable names used within the code). The names of all kinds of datatypes have to start with a capital letter. This includes classes, enumerations, structs and typedefs. The subsequent characters usually are lowercase, with the exception of new words or word parts in the identifier. Normally only letters and digits are used. Examples:

```
class Vector3
class VvcError
enum AugFileType
```

Symbolic constants consist entirely of capital letters. Words or word parts within symbolic constants are divided by the underscore character (_). Examples:

```
const int VIR_BUFFER_SIZE
#define MAX_ERRORS
```

Other identifiers start with a lowercase letter but otherwise follow the same rules as class names. This includes names for methods and all types of variables (instance variables, global variables etc.). Examples:

```
void getValue()
int curNumber
int getElement(int whichElement)
```

## Namespaces

Namespace collisions (i.e. multiple use of identical identifiers) should be avoided. In order to achieve this, a prefix indicating the current project is to be added to the beginning of all class identifiers. Examples:

```
VirError - class name in the Virtue project (Vir...)
VvcEndianStream - class name in the VVC project (Vvc...)
```

Note that this also means that all file names in the project normally start with the project prefix (e.g. VirError.h / VirError.cpp or VirColor.h / VirColor.cpp).

The project prefix should also be added to all (global) constants in the project (for instance, VIR_BUFFER_SIZE, VVC_ERROR_CODE).

Alternatively, C++ namespaces can be used for making identifiers unique. Example:

```
namespace Virtue {
    class Vector3 {
        :
    };
}
Virtue::Vector3 curPos;
```

Prefix abbrevations and namespace names should remain the same throughout the entire project. They must start with a capital letter. The rest of the letters can be upper or lower case depending on the project. Examples:

```
project:  ISConv
class:  ISCViewer
files:  ISCViewer.h, ISCViewer.cpp

project:  Virtue
class:  VirSession
files:  VirSession.h, VirSession.cpp
```

## Methods

Method names should always start with a lowercase letter. Operator overloading is to be limited to cases where the operator functionality is absolutely obvious (e.g., vector addition).

Inline methods should be defined within the class declaration using the respective C++ syntax:

```
class MyClass {
    :

    public:
        int calcValue(int a, int b) {    // keyword "inline" not
                                         // required here
            return a + b;
        }

    :
};
```

As mentioned above, private variables are to be accessed by means of get/set methods. These methods should always contain the exact name of the variable to which they provide access in their identifier. Example:

```
class MyClass2 {
    private:
        int numVertices;

    public:
        int getNumVertices();
        void setNumVertices(int newNumVertices);

    :
};
```

In the method declaration in the header file, parameter names have to be given along with the parameter types:

```
class MyClass3 {
    :
    public:
        // Wrong:  only parameter types
        void wrongMethod(int, int);

        // Correct:  parameter types AND names in declaration
        void correctMethod(int num, int versionNumber);

    :
};
```

Default parameters must be defined in the method declaration in the header file. They should **not** be defined in the implementation of the method in the source file.

```
class MyClass4 {
    :
    public:
        // Correct:  initialization in declaration
        void initExample(int num = 0, int versionNumber = 1);

    :
};

// Wrong:  initialization in implementation
void MyClass4::initExample(int num = 0, int versionNumber = 1)
{}
```

## Access Privileges

All class members must be grouped according to their access privileges. Thus *private* members (both class variables and methods) must be grouped together during declaration, as well as *protected* and *public* class members, respectively. This means that there must be only **one** block of *public* class members, **one** of *protected* (if any) and **one** of *private* ones.

```
class MyClass {
    :
    public:
        void myFirstPublicMethod();
        void mySecondPublicMethod();

    protected:
        void myFirstProtectedMethod();
        void mySecondProtectedMethod();

    private:
        myFirstPrivateMethod();
        :
        int  privateIntVar;
        char privateCharVar;
    :
};
```

Only in exceptional cases, when it is absolutely unavoidable, there can be two different blocks of the same access privilege type. One example is when using *enum* within a class. The *enum* must be a public member of the class, therefore the public group can be split in this case into two different blocks.

```
class MyClass {
    :
    public:
        enum MyEnum { ONE, TWO, THREE };
    private:
        myFirstPrivateMethod();
        :
        int  privateIntVar;
        char privateCharVar;

    public:
        myFirstPublicMethod();
        mySecondPublicMethod();
    :
};
```

The order in which the different member access groups are written is left to each developer's like.

## Project Code Documentation

Classes which are mostly finished and are not going to change significantly anymore should be documented extensively. The automatic documentation generation tool doxygen is to be used for code documentation. All code should be documented according to the documentation syntax of doxygen.

The documentation blocks should use the JavaDoc style:

```
/// Brief class description
/** This is the full description of the class DocExample...
  * :
  * :
  */
class DocExample {
    :
};
```

The "autobrief" option of doxygen should not be used, so that brief descriptions have to be given explicitly (as shown in the above example).

Within the implementation code, all parts which are complicated or non-obvious should also be documented extensively, using standard C++ comments:

```
/** <doxygen comment block>
  * :
  * :
  *
  */
int MyClass::runAlgorithm()
{
    :
    // Process all values in the input array
    for(int loop = 0; loop < inputValues.size(); loop++) {
        sum += inputValues[loop].val; // Calculate total sum for
                                      // statistics
        :
    }
    :
}
```

# Lebens- und Bildungsgang

**Jan T. Fischer**

| | |
|---|---|
| 20. Mai 1977 | geboren in Mainz |
| 1983 - 1985 | Grundschule Altusried |
| 1985 - 1987 | Grundschule Lauben |
| 1987 - 1996 | Carl-von-Linde Gymnasium Kempten, Abschluß Abitur |
| 1995 - 2000 | Tätigkeit als freiberuflicher Dozent für EDV am Kolping-Bildungszentrum Kempten |
| 1996 - 1997 | Zivildienst im Kolping-Bildungszentrum Kempten |
| 1997 - 2002 | Studium der Informatik mit den Schwerpunkten Neuroinformatik und Computer Vision (Nebenfach Wirtschaftswissenschaften) an der Universität Ulm |
| 1998 - 2001 | Wissenschaftliche Hilfskraft in Lehre und Forschung in den Bereichen Praktische Informatik, Neuroinformatik, Robotik und Computergraphik an der Universität Ulm |
| 2000 - 2001 | Werkstudent am Virtual Reality Competence Center der Daimler-Chrysler AG in Ulm |
| 2001 - 2002 | Freiberuflicher Dozent für Java-Programmierung in der Fachinformatiker-Ausbilung am Kolping-Bildungszentrum Memmingen |
| 2002 | Diplomarbeit *Interaktive Spezifikation von Domänen und Detektion partieller dynamischer Verdeckungen in Augmented-Reality Umgebungen* am Virtual Reality Competence Center der DaimlerChrysler AG in Ulm |
| 07 / 2002 | Abschluß des Studiums als Diplom-Informatiker |
| seit 11 / 2002 | Wissenschaftlicher Mitarbeiter in der Arbeitsgruppe Visual Computing for Medicine (PD Dr. Dirk Bartz) am Lehrstuhl für Graphisch-Interaktive Systeme von Prof. Dr. Dr. Wolfgang Straßer des Wilhelm-Schickard-Instituts für Informatik der Universität Tübingen |