Bridging Multiple User Interface Dimensions with Augmented Reality

Dieter Schmalstieg
Vienna University of Technology, Austria
dieter@cg.tuwien.ac.at

Anton Fuhrmann
Research Center for Virtual Reality and Visualization, Vienna, Austria
fuhrmann@vrvis.at

Gerd Hesina
Vienna University of Technology, Austria
hesina@cg.tuwien.ac.at

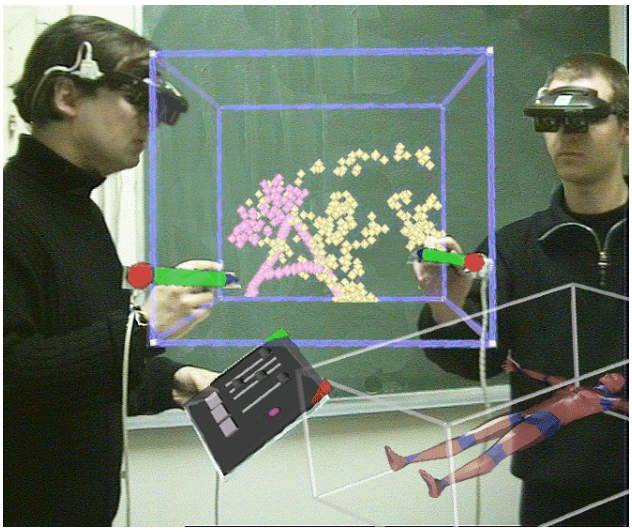

Figure 1: Collaborative work in *Studierstube*: 3D painting application window (with focus, middle) and object viewer window (without focus, lower right)

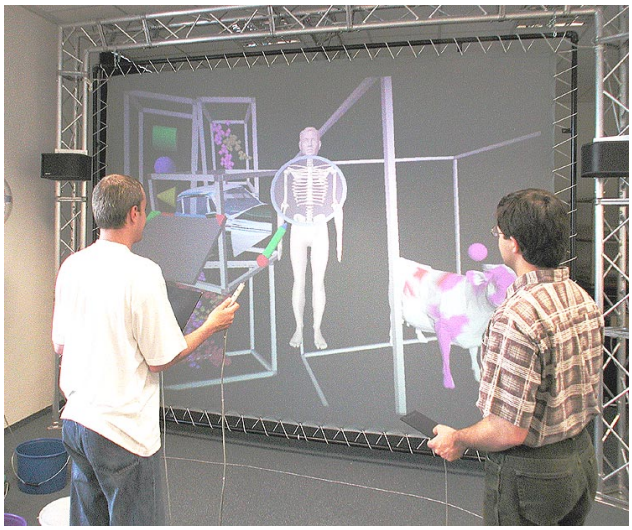

Figure 2: Two *Studierstube* users working jointly on multiple applications in front of a large screen, usually with passive stereo glasses (not shown)

Abstract

Studierstube is an experimental user interface system, which uses collaborative augmented reality to incorporate true 3D interaction into a productivity environment. This concept is extended to bridge multiple user interface dimensions by including multiple users, multiple host platforms, multiple display types, multiple concurrent applications, and a multi-context (i. e., 3D document) interface into a heterogeneous distributed environment. With this architecture, we can explore the user interface design space between pure augmented reality and the popular ubiquitous computing paradigm. We report on our design philosophy centered around the notion of contexts and locales, as well as the underlying software and hardware architecture. Contexts encapsulate a live application together with 3D (visual) and other data, while locales are used to organize geometric reference systems. By separating geometric relationships (locales) from semantic relationships (contexts), we achieve a great amount of flexibility in the configuration of displays. To illustrate our claims, we present several applications including a cinematographic design tool which showcases many features of our system.

1. Introduction

Technical progress in recent years gives reason to believe that virtual reality (VR) has a good potential as a user interface of the future. At the moment, VR applications are usually tailored to the needs of a very specific domain, such as a theme park ride or virtual mock-up. We believe that augmented reality (AR), the less obtrusive cousin of VR, has a better chance to become a viable user interface for everyday productivity applications, where a large variety of tasks has to be covered by a single system. Rather than forcing a user to deal exclusively with a virtual environment, less rigid approaches like UNC's *office of the future* [12] embed VR and AR tools in a conventional work environment.

In a more general sense, this principle is known as *ubiquitous computing* [22], describing a world where computers are embedded in large numbers in our everyday surrounding, allowing constant access to networked resources. Some researcher argue that AR is the opposite of ubiquitous computing, because users carry their computing devices – such as head-mounted (HMDs) displays - to the places they go to rather than expecting the computing devices to be there already.

We believe that these two concepts are just extremes on a scale, and that a lot of useful user interface concepts can be found in between. Our current work on the *Studierstube* project, which started as a pure augmented reality setup [15], focuses on experimenting with the possibilities of new user interfaces that incorporate AR. For efficient experimentation, we have implemented a toolkit that generalizes over multiple user interface dimensions, allowing rapid prototyping of different user interface styles. The *Studierstube* user interface spans the following dimensions:

1.1. Multiple users

The system allows multiple users to collaborate (Figure 1, Figure 2). While we are most interested in computer-supported face-to-face collaboration, this definition also encompasses remote collaboration. Collaboration of multiple users implies that the system will typically incorporate *multiple host computers*. However, we also allow multiple users to interface with a single host (e.g. via a large screen display), and a single user to interface with multiple computers at once. On a very fundamental level, this means that we are dealing with a distributed system. It also implies that *multiple types of output devices* such as HMDs, projection-based displays, hand-held displays etc. can be handled and that the system can span *multiple operating systems*.

1.2. Multiple contexts

Contexts are the fundamental units from which the *Studierstube* environment is composed. A context is a union of *data* itself, the data's *representation* and an *application* which operates on the data. Contexts are thus structured along the lines of the model-view-controller (MVC) paradigm known from Smalltalk's windowing system [6]: *Studierstube*'s data, representation, and application correspond to MVC's model, view, and controller, respectively. Not surprisingly, this structure makes it straightforward to generalize established properties of 2D user interfaces to three dimensions.

In other words, a context encapsulates visible and invisible application-specific data together with the responsible application. The notion of an application is therefore completely hidden from the user, in particular, users never have to "start" an application, they simply open a context of a specific type. Compared to the desktop metaphor, this approach is much closer to the concept of an *information appliance*, which is always "on" (compare [2]).

In a conventional desktop system, the data representation of a document is typically a single 2D window. Analogously, in our three-dimensional user interface, we define a context's representation as a three-dimensional structure contained in a certain volume – a *3D-window*. Unlike its 2D counterpart, a context can be shared by any group of users, and even more importantly, can be present in *multiple locales* simultaneously by replication.

Every context is an instance of a particular application type. Contexts of different types can exist concurrently, which results in *multi-tasking* of *multiple applications*, a feature which is well established within the desktop metaphor, but rarely implemented in virtual environments. Moreover, *Studierstube* also allows multiple contexts of the same type to co-exist, allowing a single application to work with multiple data sets. In the desktop metaphor, this feature is generally known as a *multiple document interface*. Note that it differs from simply allowing multiple instances of the same application which are unaware of each other. Multiple contexts of the same type are aware of each other can share features and data. For example, consider the shared "slide sorter" from section 5.

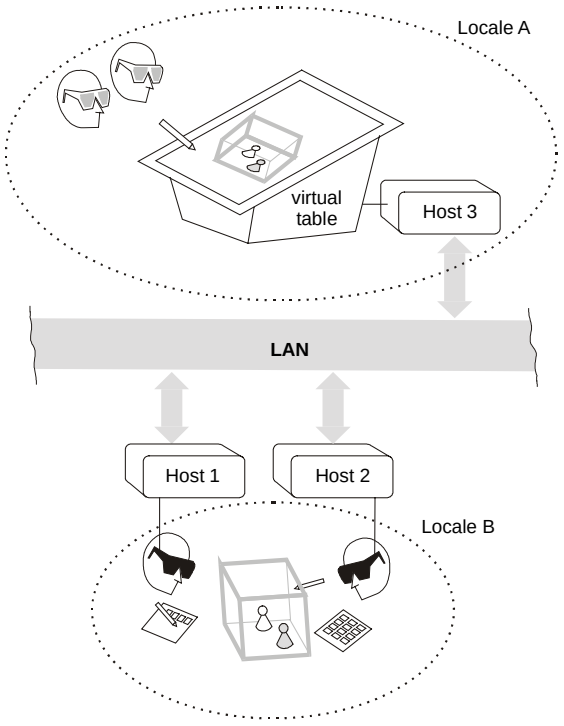

Figure 3: Multiple locales can simultaneously exist in *Studierstube*. They can be used to configure different output devices and to support remote collaboration

1.3. Multiple locales

Locales correspond to coordinate systems in the virtual environment. They usually coincide with physical places (such as a lab or conference room, or parts of rooms), but they can also be portable and associated with a user, or used arbitrarily – we even allow (and use) overlapping locales in the same physical space. We define that every display used in a *Studierstube* environment shows the

content of exactly one locale. Every context can (but need not) be replicated in every locale; these replicas will be kept synchronized by *Studierstube's* distribution mechanism.

To understand why the separation of locales and contexts is necessary, consider the following examples:

- Multiple users are working on separate hosts. They can share contexts, but can layout the context representations (3D-windows) arbitrarily according to screen format and personal preferences. This is made possible by defining separate locales, as the position of 3D-windows is not shared across locale boundaries (Figure 3). The hosts can be in separate buildings for remote collaboration, or they can be placed side by side. In the latter case, locales would probably overlap, as users might see several or all screens.
- A user wearing a see-through HMD is looking at a large projection screen *through* the HMD. Both display devices (HMD, projection screen) can be set to use the same locale, so the graphics in a user's HMD may *augment* the projection screen's output. Of course this setup is view-dependent and works for only one user, so alternatively, the projection screen may use a separate locale, and present graphical elements which are complementary to the HMD output.

By separating locales (geometric relationships) from contexts (semantic relationships), we achieve a great amount of flexibility in the configuration of displays. This not only allows to connect multiple *Studierstube* environments over a network for remote collaboration, but also to set up an environment with multiple co-located, i. e., overlapping locales. Consider as a scenario a spacecraft mission control center with dozens of collaborating operators assembled in a large hall. Every involved user will assume a specific role and require specific tools and data sets, while some aspects of the mission will be shared by all users. A naïve approach of embedding all users in a single locale means that users in close proximity can work in a shared virtual space, while other users who desire to participate are too far away to see the data well, and are not within arm's reach for manual interaction. By separating contexts from locales, a remote user can import the context into a separate locale, and interact with it conveniently. While our available resources do not allow us to verify such large-scale interaction, in section 5 we present some results that back up our considerations.

The system presented in this paper must be understood as an experimental platform for exploring the design space that emerges from bridging multiple user interface dimensions. It can neither compete in maturity and usability with the universally adopted desktop metaphor nor with more streamlined, specialized virtual environment solutions (e. g., CAVEs). However, *Studierstube* demonstrates a design approach for next generation user interfaces as well as solutions on how to implement these interfaces.

2. Previous work

Almost a decade ago, Weiser introduced the concept of *ubiquitous computing* as a future paradigm on interaction with computers [22]. In his vision, computers are constantly available in our surrounding by embedding them into everyday items, making access to information almost transparent. In contrast, *augmented reality* systems focus on the use of personal displays (such as see-through head-mounted displays) to enhance a user's perception by overlaying computer generated images onto a user's view of the real-world.

Collaborative augmented reality enhances AR with distributed system support for multiple users with multiple display devices, allowing a co-located joint experience of virtual objects [3, 15]. Some researchers are experimenting with a combination of collaborative AR, ubiquitous computing and other user interface concepts. Prominent examples include EMMIE developed at Columbia University [4, 8], work by Rekimoto [13], and the Tangible Bits Project at MIT [9, 21]. These systems share many aspects with our approach for a collaborative augmented reality system making use of a variety of stationary as well as portable devices.

Working with such a system will inevitably require transfer of data from one computer's domain to another. For that aim, Rekimoto [14] proposes *multi-computer direct manipulation*, i. e. drag and drop across system and display boundaries. To implement this approach, a physical prop (in Rekimoto's case, a pen) is used as a virtual "store" for the data, while in reality the data transfer is carried out via the network using the pen only as a passive locator. Similar transfer functions are available in EMMIE [4]. Such use of passive objects as perceived media containers is also implemented by the Tangible Bits group's mediaBlocks [21].

Other sources of inspiration for multiple dimensions in 3D user interfaces are CRYSTAL [20], which allows concurrent execution of multiple applications in the same three-dimensional workspace, and SPLINE [1], which introduced the concept of multiple *locales* within one large virtual environment. Note that unlike SPLINE, *Studierstube's* allows multiple locales to overlap.

3. Background: *Studierstube*

The original *Studierstube* architecture [15, 18] was a collaborative augmented reality system allowing multiple users to gather in a room and experience the sensation of a shared virtual space that can be populated with three-dimensional data. Head-tracked see-through head-

mounted displays (HMDs) allow each user to choose an individual viewpoint while retaining full stereoscopic graphics.

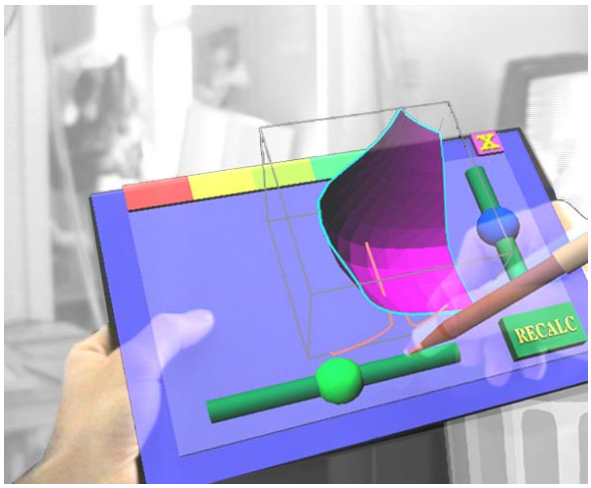

Figure 4: The Personal Interaction Panel combines tactile feedback from physical props with overlaid graphics to form a two-handed general purpose interaction tool.

The personal interaction panel (PIP), a two-handed interface composed of pen and pad, both fitted with magnetic trackers, is used to control the application [19]. It allows the straightforward integration of conventional 2D interface elements like buttons, sliders, dials etc. as well as novel 3D interaction widgets (Figure 4). The haptic feedback from the physical props guides the user when interacting with the PIP, while the overlaid graphics allows the props to be used as multi-function tools. Every application may display its own interface in the form of a PIP ”sheet”, which appears on the PIP when the application is in focus. The pen and pad are our primary interaction devices.

While the original *Studierstube* architecture from [18] incorporated simple distribution mechanisms to provide graphics from multiple host computers and shared data from a separate device (tracker) server, the initial networking approach later turned out to be insufficient for the evolving distribution requirements. An even more limiting factor was that the toolkit allowed to run only a single application and a single context at a time. Our efforts towards a follow-up version resulted in support for projection-based platforms [16] and a toolkit for distributed graphics [7]. This paper presents the results of a two-year long redesign process of *Studierstube* incorporating all these features into a new framework.

4. Implementation

Our software development environment is realized as a collection of C++ classes built on top of the Open Inventor (OIV) toolkit [17]. The rich graphical environment of OIV allows rapid prototyping of new interaction styles. The file format of OIV enables convenient scripting, overcoming many of the shortcomings of compiled languages without compromising performance. At the core of OIV is an object-oriented scene graph storing both geometric information and active interaction objects. Our implementation approach has been to extend OIV as needed, while staying within OIV’s strong design philosophy.

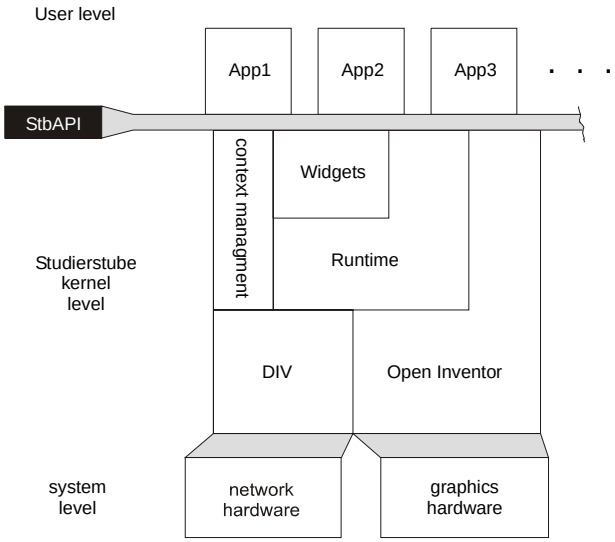

Figure 5: The *Studierstube* software is composed of an interaction toolkit and runtime system. The latter is responsible for managing context and distribution.

This has lead to the development of two intertwined components: A toolkit of extensions of the OIV class hierarchy (mostly interaction widgets capable of responding to 3D events), and a runtime framework which provides the necessary environment for *Studierstube* applications to execute (Figure 5). Together, these components form a well-defined application programmer’s interface (API), which extends the OIV API, and also offers a convenient programming model to the application programmer (section 4.4). Applications are written and compiled as separate shared objects (.so for IRIX, .dll for Win32), and dynamically loaded into the runtime framework. A safeguard mechanism makes sure only one instance of each application is loaded into the system at any time. Besides decoupling application development from system development, dynamic loading of objects also simplifies distribution as application components can be loaded by each host whenever needed. All these features are not unique to *Studierstube*, but rarely found in virtual environment software.

By using this dynamic loading mechanism, *Studierstube* supports multi-tasking of *different*

applications (e.g. a painting application and a 3D modeler), but also multiple concurrent contexts associated with the *same* application (Figure 6). This approach is similar to popular desktop systems such as the *multiple document interface*.

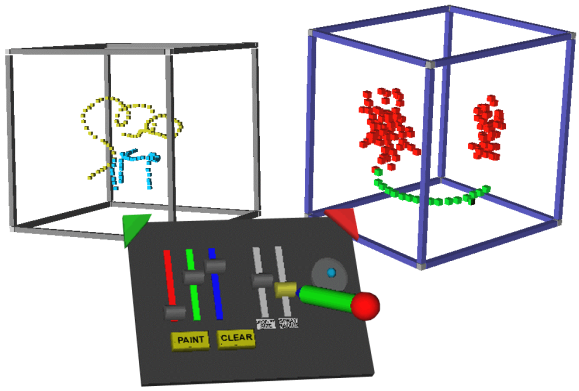

Figure 6: Multiple document interface in 3D – the right window has the user's focus and can be manipulated with the current PIP sheet.

Depending on the semantics of the associated application, ownership of a context may or may not privilege a user to perform certain operations on the information (such as object deletion). Per default, users present in the same locale will share a context. A context – represented by its 3D-window - is owned by one user, and subscribed by others. Per default, a context is visible to all users and can be manipulated by any user in the locale.

4.1. 3D-windows

The use of windows as abstraction and interaction metaphor is a long-time convention in 2D GUIs. Its extension to three dimensions seems logical [5, 20] and can be achieved in a straightforward manner: Using a box instead of a rectangle seems to be the easiest way of preserving the well-known properties of desktop windows when migrating into a virtual environment. It supplies the user with the same means of positioning and resizing the display area and also defines its exact boundaries.

A context is normally represented in the scene by a 3D-window, although we allow a context to span multiple windows. The 3D-window class is a container associated with a user-specified scene graph. This scene graph is normally rendered with clipping planes set to the faces of the containing box, so that the content of the window does not protrude from the window's volume. Nested windows are possible, although we have found little use for them. The window is normally rendered with associated "decoration" that visually defines the windows extent and allows it to be manipulated with the pen (move, resize etc). The color of the decoration also indicates whether a window has a user's focus (and hence receives 3D event from that user). Like their 2D counterparts, 3D-windows can be minimized (replaced by a three-dimensional icon to save space in a cluttered display), and maximized (scaled to fill the whole work volume and receive input events exclusively). Typically, multiple context of the same type will maintain structurally similar windows, but this decision is at the discretion of the application programmer.

4.2. PIP sheets

Studierstube applications are controlled either via direct manipulation of the data presented in 3D-windows, or via a mixture of 2D and 3D widgets on the PIP. A set of controls on the PIP – a *PIP sheet* - is implemented as an OIV scene graph composed primarily of *Studierstube* interaction widgets (such as buttons etc.). However, the scene graph may also contain geometry (e. g., 2D and 3D icons) that are useful to convey user interface state or merely as decoration.

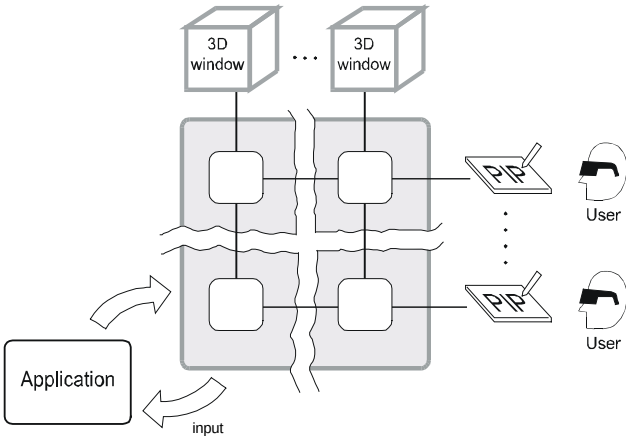

Figure 7: Multiplicity relationships in *Studierstube* - control elements on the PIP are instantiated separately for every (user, 3D-window) pair

Every type of context defines a PIP sheet template, a kind of application resource. For every context and user, a separate PIP sheet is instantiated. Each interaction widget on the PIP sheet can therefore have a separate state. For example, the current paint color in our artistic spraying application (Figure 6) can be set individually by every user for every context. However, widgets can also be shared by all users, all contexts, or both. Consequently, *Studierstube's* 3D event routing involves a kind of multiplexer between windows and users' PIP sheets (Figure 7).

4.3. Distributed execution

The distribution of *Studierstube* requires that for each replica of a context all graphical and application-specific data is locally available at each host which has a replica.

In general, applications written with OIV encode all relevant information in the scene graph, so replicating the scene graph at each participating host already solves most of the problem.

For that aim, we have created Distributed Open Inventor (DIV) [7] as an extension (more a kind of plug-in) to OIV. The DIV toolkit extends OIV with the concept of a distributed shared scene graph, similar to distributed shared memory. From the application programmer's perspective, multiple workstations share a common scene graph. Any operation applied to a part of the shared scene graph will be reflected by the other participating hosts. All this happens to the application programmer in an almost completely transparent manner by capturing and distributing OIV's *notification events*. A scene graph need not be totally replicated – local variations (compare [10]) in the scene graph can be introduced, which is among others useful for fine-tuning low-latency operations such as dragging.

More importantly, local variations allow us to resolve distribution on a *per-context* base. A context is owned by one workstation (called a master context), which will be responsible of processing all relevant interaction on the application, while other workstations (in the same locale and in other locales) may replicate the context (as a slave context).

The roles that contexts may assume (master or slave) affect the status of the context's application part. The context data and its representation (window, PIP sheet etc.) stay synchronized over the whole lifespan of the context for every replica. The application part of a master context is active and modifies context data directly according to the users' input. A slave context's application is dormant and does not react to user input (for example, no callbacks are executed if widgets are triggered). Instead, a slave context relies on updates to be transmitted via DIV. Note that context replicas can swap roles (e. g., by moving master contexts to achieve load balancing), but at any time there may only be one master copy per replicated context.

The replication on a per context-base provides coarse-grained parallelism. At the same time the programming model stays simple, as the programmer is spared to solve difficult concurrency issues and all relevant input can be processed in a single address space.

Once the low-level replication of context data is taken care of by DIV, the high-level context management protocol is fairly simple: A dedicated session manager process serves as a mediator among hosts as well as a known point of contact for newcomers. The session manager does not have a heavy workload compared to the hosts running the *Studierstube* user interface, but its directory services are essential. For example, it maintains a list of all active hosts and which contexts they own or subscribe, it gets to decide about policy issues such as load balancing etc.

Finally, input is managed separately by dedicated device servers (typically PCs running Linux), which also perform the necessary filtering and prediction. The tracker data is then multicast in the LAN, so it is simultaneously available to all hosts for rendering.

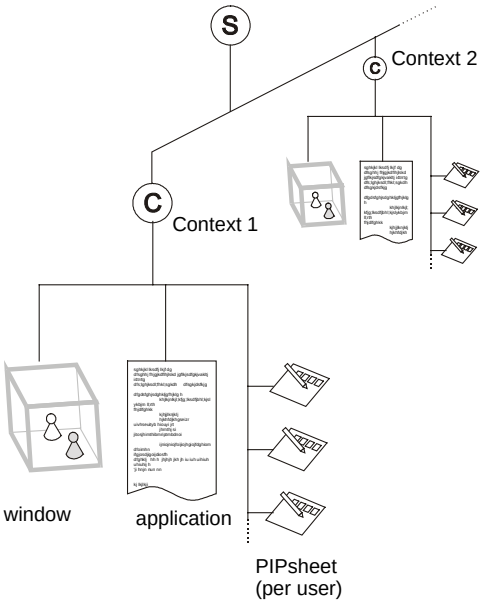

Figure 8: A context is implemented as a node in the scene graph, as are windows and pip sheets. This allows to organize all relevant data in the system in a single hierarchical data structure.

4.4. Application programmer's interface

The *Studierstube* API imposes a certain programming model on applications, which is embedded in a foundation class, from which all *Studierstube* applications are derived. By overloading certain polymorphic methods of the foundation class, a programmer can customize the behavior of the application. The structure imposed by the foundation class makes sure the application allows multiple contexts to be created (i. e., offers the equivalent to a multiple document interface), each of which can be operated in both master mode (normal application processing) and slave mode (same data model, but all changes occur remotely through DIV).

The key to achieve all this is to make the context itself a *node* in the scene graph. Such context nodes are implemented as OIV *kit* classes. Kits are special nodes that can store both fields, i. e., simple attributes, and child nodes, both of which will be considered part of the scene graph and thus implicitly be distributed by DIV. Default parts of every context are at least one 3D-window node, which is itself an OIV kit and contains the context's "client area" scene graph, and an array of PIP sheets,

which are also special scene graphs. In other words, data, representation, and application are all embedded in a single scene graph (Figure 8), which can be conveniently managed by the *Studierstube* framework.

To create a useful application with all the properties mentioned above, a programmer need only create a subclass of the foundation class and overload the 3D-window and PIP sheet creation methods to return custom scene graphs. Typically, most of the remaining application code will consist of *callback* methods responding to certain 3D events such as button press or 3D direct manipulation events. Although the programmer has great freedom to use anything that the OIV and *Studierstube* toolkits offer, it is a requirement that any instance data is stored in the derived context class as a field or node, or otherwise it will not be distributed. However, this is not a restriction in practice, as all basic data types are available in both scalar and vector format as fields, and new types can be created should the existing ones turn out to be insufficient (a situation that has not occurred to us yet).

Note that allowing a context to operate in both master and slave mode has implications on how contexts can be distributed: It is not necessary to store all master contexts of a particular type at one host. Some master contexts may reside on one host, some on another host – in that case, there will be corresponding slave contexts at the respective other host, which are also instances of the same kit class, but initialized to function as slaves. In essence, our API provides a distributed multiple document interface.

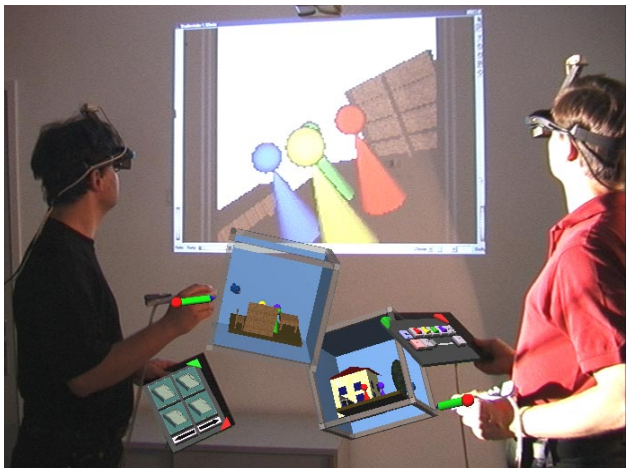

Figure 9: Storyboard application with two users and two contexts as seen from a third "virtual" user used for video documentation. In the background the video projection is visible.

5. Results

To demonstrate our framework, we chose the application scenario of *Storyboard design*. This application is a prototype of a cinematic design tool. It allows multiple users to concurrently work on a storyboard for a movie or drama. Individual scenes are represented by their stage sets, a kind of *world in miniature* [11].

Every scene is represented by its own context, and embedded in a 3D-window. Users can manipulate the position of props in the scene as well as the number and placement of actors (represented by colored board game figures), and finally the position of the camera (Figure 9, Figure 10).

All contexts share an additional large *slide show* window, which shows a 2D image of the selected scene from the current camera position. By flipping through the scenes in the given sequence, the resulting slide show conveys the visual composition of the movie.

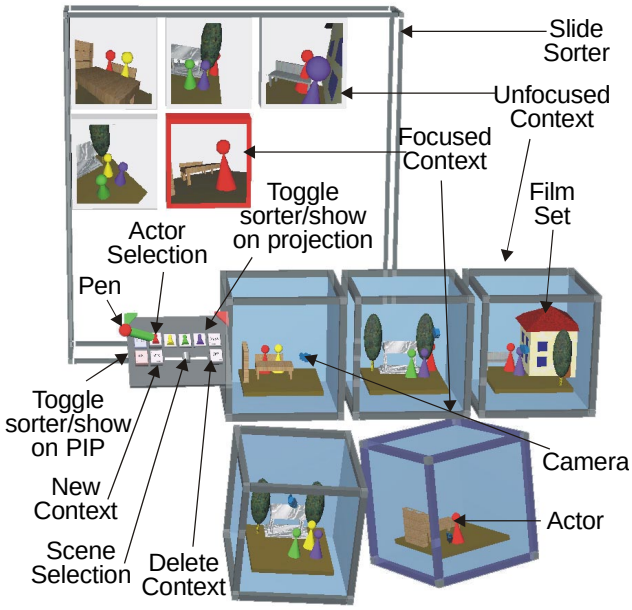

Figure 10: The Storyboarding application allows the 3D placement of actors, props, and cameras. The slide sorter shows a storyboard of all camera "shots"

Alternatively, a user may change the slide show to a "slide sorter" view inspired by current presentation graphics tools, where each scene is represented by a smaller 2D image, and the sequence can be rearranged by simple drag and drop operations. The slide sorter comes closest to the traditional *storyboard* used in cinematography. It appears on the PIP for easy manipulation as well as on the larger projection screen.

Using the distributed *Studierstube* framework, we ran the Storyboard application in different configurations.

5.1. Heterogeneous displays

Our first configuration (Figure 9, Figure 11) consisted of three hosts (SGI Indigo2, Intergraph TZ1 Wildcat, SGI O2), two users, and two *locales* (Figure 12). It was designed to show the convergence of multiple users (real

ones as well as virtual ones), contexts, locales, 3D-windows, hosts, displays and operating systems.

The two users were wearing HMDs, both connected to the Indigo2's multi-channel output, and seeing head-tracked stereoscopic graphics. They were also fitted with a pen and pad each. The Intergraph workstation was driving an LCD video projector to generate a monoscopic image of the projection screen (without viewpoint tracking) on a projection wall. The slider show/sorter 3D-window was hidden from graphics output on the HMDs, so the users could see the result of their manipulation of the miniature scenes on the large bright projection exploiting the see-through capability of the HMDs.

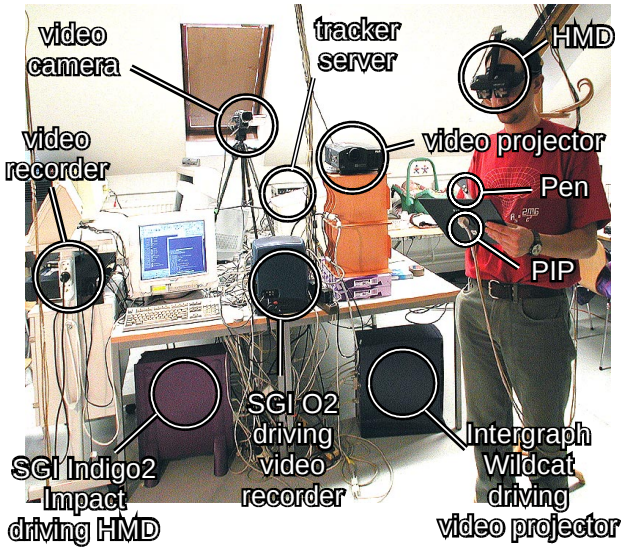

Figure 11: Hardware setup for the heterogeneous display experiment

Users were able to perform some private editing on their local contexts, then update the slide show/sorter to discuss the results. Typically, each user would work on his or her own set of scenes. However, we choose to make all contexts visible to both users, so collaborative work on a single scene was also possible. The slide sorter view was shared between both users, so global changes to the order of scenes in the movie were immediately recognizable.

The third host – the O2 – was configured to combine the graphical output (monoscopic) from *Studierstube* with a live video texture obtained from a video camera pointed at the users and projection screen. The O2 was configured to render for a virtual user, whose position was identical with the physical camera. This feature was used to document the system on video. The configuration used two locales, one shared by the two users and the O2, while a separate locale was used for the Intergraph driving the projection screen (again viewed by a virtual user). The additional video host allowed us to perform live composition of the users' physical and virtual actions on video, while the video projector driving the projection screen could be freely repositioned without affecting the remainder of the system (Figure 12).

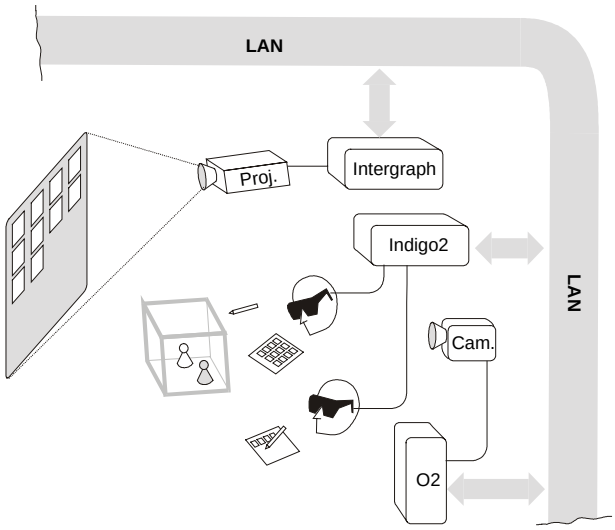

Figure 12: Heterogeneous displays – two users simultaneously see shared graphics (via their see-through HMDs) and a large screen projection

5.2. Symmetric workspace

The second example was intended to show multi-user collaboration in pure augmented reality with multiple hosts. The Storyboarding application was executed in a more conventional augmented reality setup consisting of two hosts (Indigo2, Intergraph), two users, and one *locale* (Figure 13). Both users were wearing HMDs again, but the first user was connected to the Indigo2, while the second user was connected to the Intergraph. In this configuration, the slide show/sorter was included in the graphics shown via the HMD rather than projected by a separate video projector.

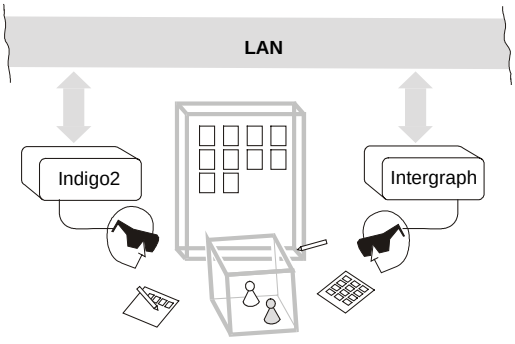

Figure 13: A symmetric workspace configuration uses homogeneous displays (2 HMDs) to present a shared environment to multiple users in one locale

While the obtainable frame rate was significantly higher than for the first configuration, since rendering load for

the two users was distributed over two hosts, no high resolution wide field-of-view projection was available for the slide show/sorter. Consequently, only one locale was necessary since users shared the same physical space.

5.3. Remote collaboration

The third example was created to show remote collaboration of multiple users. In this setup, we built a second *Studierstube* environment in the laboratory next door to experiment with the possibilities of remote collaboration. We then let two users collaborate remotely using the Storyboard application.

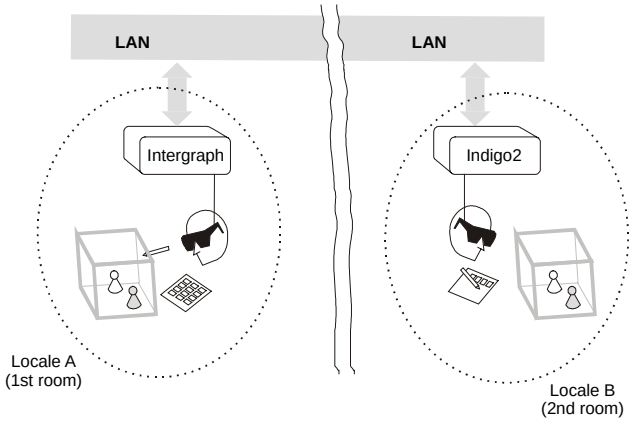

Figure 14: Remote collaboration: Two geographically separated users experience a shared environment

Note that the results are preliminary in the sense that all hosts were connected to the same LAN segment, and network performance is thus not representative of what one would get over a wide area network connection. However, this was not the current focus of investigation.

The system consisted of two hosts (Intergraph in the first laboratory, Indigo2 in the second), two users and two locales (Figure 14). Each user was wearing a HMD connected to the local workstation. In contrast to configuration from section 5.2, two locales were used as the users did not share a physical presence. The sharing of context, but not locale, allowed them to rearrange their personal workspace at their convenience without affecting collaboration.

5.4. Multiple applications

Finally, Figure 1 and Figure 2 show users working with multiple applications such as spraying, painting, and object viewing tools on two possible platforms: HMDs and a large polarized stereo projection wall.

6. Discussion

As observed by Tsao and Lumsden [20], in order to be successful for everyday productivity work situations, virtual environment systems must allow "multi-tasking" and "multi-context" operation. By multi-tasking they mean that the virtual environment can be re-configured to execute a particular application, i. e., there is a separation of VR system software and application software.

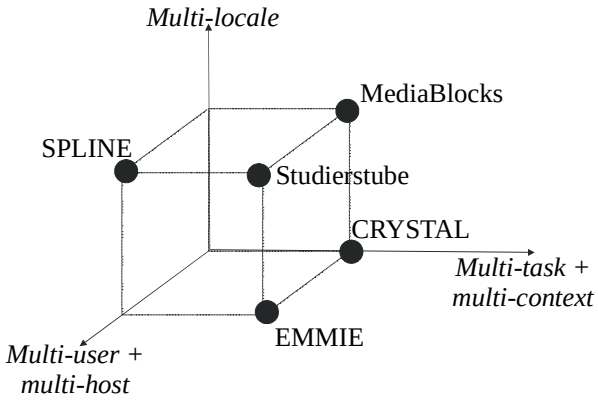

Figure 15: Extended taxonomy for multiple dimensions of user interfaces with some related work (adapted from CRYSTAL).

Multi-context operation goes beyond that by allowing multiple applications to execute concurrently rather than sequentially. They also point out that this resembles a development earlier experienced for 2D user interfaces, which evolved from single-application text consoles to multi-application windowing systems. It is no surprise that by "judicious borrowing", many useful results from 2D user interfaces become applicable to 3D, as is evident with *Studierstube's* PIP, 3D-windows, or 3D event system.

However, the CRYSTAL system from [20] does not incorporate true multi-user operation, and consequently has no need for multiple locales. Extending the taxonomy from CRYSTAL, Figure 15 compares some relevant work. For example, MIT's mediaBlocks [21] allow a user to work with different manipulators, which are dedicated devices for specific applications, and the mediaBlocks themselves are a very elegant embedding for context data. However, although principally possible, no multi-user scenarios were demonstrated.

In contrast, SPLINE [1] is designed towards multi-user interaction. While SPLINE completely immerses a user in a purely virtual world and thus does not meet our definition of a work environment, it features multiple locales that correspond to activities (for example, chat takes place in a street café, while train rides take place on a train).

The closest relative to our work is Columbia's EMMIE [4]. Except for explicit support of locales, EMMIE shares many basic intentions with our research, in particular concurrent use of heterogeneous media in a collaborative work environment. Like ourselves, the authors of EMMIE believe that future user interfaces will require a broader design approach integrating multiple user interface

dimensions before a successor to the desktop metaphor can emerge.

7. Conclusions and future work

We have presented *Studierstube*, a prototype user interface that uses collaborative augmented reality to bridge multiple user interface dimensions: Multiple users, context, and locales as well as applications, 3D-windows, hosts, display platforms, and operating systems. *Studierstube* supports collaborative work by coordinating a heterogeneous distributed system based on a distributed shared scene graph and a 3D interaction toolkit. This architecture allows to combine multiple approaches to user interfaces as needed, so that it becomes easy to create a 3D work environment, which can be personalized, but also lends itself to computer supported cooperative work.

Our implementation prototype shows that despite its apparent complexity, such a design approach is principally feasible, although much is left to be desired in terms of quality and maturity of hard- and software. However, addressing issues such as display update rate and tracking accuracy is out of scope of this work.

Our future interest will focus on bringing the element of mobility into the *Studierstube* environment. While the name *Studierstube* ("study room") may be no longer appropriate, we envision a portable 3D information space that allows ad-hoc networking for instant collaboration of augmented users. Our goal is to allow users to take 3D contexts "on the road" and even dock into a geographically separate environment without having to shut down live applications.

Acknowledgments

This project was sponsored by the Austrian Science Fund *FWF* under contract no. P-12074-MAT. Special thanks to Markus Krutz, Rainer Splechtna, Hermann Wurnig, and Andreas Zajic for their contributions to the implementation, to Zsolt Szalavári and Michael Gervautz for inventing the PIP, and to M. Eduard Gröller for his spiritual guidance.

Web information

http://www.cg.tuwien.ac.at/research/vr/studierstube/

References

1. Barrus, J., R. Waters, R. Anderson. Locales and Beacons: Precise and Efficient Support for Large Multi-User Virtual Environments. *Proc. VRAIS '96*, pp. 204-213, 1996.
2. Billinghurst M., J. Bowskill, M. Jessop, J. Morphett. A Wearable Spatial Conferencing Space, *Proc. ISWC '98*, pp. 76-83, 1998.
3. Billinghurst M., S. Weghorst, T. Furness III: Shared Space: An Augmented Reality Approach for Computer Supported Collaborative Work, *Virtual Reality: Virtual Reality - Systems, Development and Applications*, 3(1), pp. 25-36, 1998.
4. Butz A., T. Höllerer, S. Feiner, B. MacIntyre, C. Beshers. Enveloping Computers and Users in a Collaborative 3D Augmented Reality, *Proc. IWAR '99*, pp. 1999.
5. Feiner S., C. Beshers. Worlds Within Worlds: Metaphors for Exploring N-Dimensional Virtual Worlds, *Proc. UIST '90*, pp. 76-83, 1990.
6. Goldberg A., D. Robson. *Smalltalk-80: The language and its implementation*. Addison-Wesley, Reading MA, 1983.
7. Hesina G., D. Schmalstieg, A. Fuhrmann, W. Purgathofer. Distributed Open Inventor: A Practical Approach to Distributed 3D Graphics, *Proc. VRST '99*, London, pp. 74-81, Dec. 1999.
8. Höllerer T., S. Feiner, T. Terauchi, G. Rashid, D. Hallaway. Exploring MARS: Developing indoor and outdoor user interfaces to a mobile augmented reality systems, *Computers & Graphics*, 23(6), pp. 779-785, 1999.
9. Ishii H., B. Ulmer. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms, *Proc. CHI '97*, pp. 234-241, 1997.
10. MacIntyre B., S. Feiner. A Distributed 3D Graphics Library, *Proc. SIGGRAPH '98*, pp. 361-370, 1998.
11. Pausch R., T. Burnette, D. Brockway, M. Weiblen. Navigation and Locomotion in Virtual Worlds via Flight into Hand-Held Miniatures, *Proc. SIGGRAPH '95*, pp. 399-401, 1995.
12. Raskar R., G. Welch, M. Cutts, A. Lake, L. Stesin, H. Fuchs. The office of the future: A unified approach to image-based modeling and spatially immersive displays, *Proc. SIGGRAPH '98*, pp. 179-188, 1998.
13. Rekimoto J. A Multiple Device Approach for Supporting Whiteboard-based Interactions, *Proc. CHI '98*, pp. 344-351, 1998.
14. Rekimoto J. Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments, *Proc. UIST '97*, pp. 31-39, 1997.
15. Schmalstieg D., A. Fuhrmann, Zs. Szalavari, M. Gervautz. Studierstube - Collaborative Augmented Reality, *Proc. Collaborative Virtual Environments '96*, Nottingham, UK, Sep. 1996.
16. Schmalstieg D., L. M. Encarnação, Zs. Szalavári. Using Transparent Props For Interaction With The Virtual Table, *Proc. SIGGRAPH Symp. on Interactive 3D Graphics '99*, pp. 147-154, Atlanta, GI, April 1999.
17. Strauss P., R. Carey. An object oriented 3D graphics toolkit, *Proc. SIGGRAPH '92*, pp. 341-347, 1992.
18. Szalavári Zs., A. Fuhrmann, D. Schmalstieg, M. Gervautz. *Studierstube* - An Environment for Collaboration in Augmented Reality, *Virtual Reality - Systems, Development and Applications*, 3(1), pp. 37-49, 1998.
19. Szalavári Zs., M. Gervautz. The Personal Interaction Panel - A Two-Handed Interface for Augmented Reality, *Computer Graphics Forum*, 16(3), pp. 335-346, Sep. 1997.
20. Tsao J., C. Lumsden. CRYSTAL: Building Multicontext Virtual Environments, *Presence*, 6(1), pp. 57-72, 1997.
21. Ullmer B., H. Ishii, D. Glas. mediaBlocks: Physical Containers, Transports, and Controls for Online Media, *Proc. SIGGRAPH '98*, pp. 379-386, July 1998.
22. Weiser M. The Computer for the twenty-first century. *Scientific American*, pp. 94-104, 1991.