Diploma Thesis

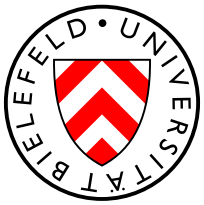
Multimodal Augmented Reality to Enhance Human Communication

Christian Mertes

31 August 2008

Supervised by

Dr. Thomas Hermann
Dr.-Ing. Marc Hanheide
Dipl. Inform. Angelika Dierker
Prof. Dr.-Ing. Gerhard Sagerer



Multimodal Augmented Reality to Enhance Human Communication

Christian Mertes

31 August 2008

Diploma Thesis submitted to the
Faculty of Technology of Bielefeld University

Abstract

Humans naturally use an impressive variety of ways to communicate. In this work, we will investigate the possibilities of complementing these natural communication channels with artificial ones. For this, augmented reality is used as a technique to add synthetic visual and auditory stimuli to people's perception. A system for the mutual display of the gaze direction of two interactants is presented and its acceptance is shown through a study. Finally, future possibilities of promoting this novel concept of artificial communication channels are explored.

A Note on the Use of Pronouns

For the sake of readability, I will exclusively use male pronouns when referring to individuals who may as well be female as they may be male. I believe, in a work like this, cognitive capacity is better spent on understanding a sentence's meaning than on the also important reminder of the falseness and injustice of male prototypes forming mental representations of descriptions using this admittedly unfortunate and biasing diction.

Nevertheless, I apologize to anyone who does not accept this explanation as satisfactory and might feel offended or segregated.

Contents

List of Figures

1 Introduction

In this first chapter I will write some introductory words on the idea behind this work, sum up very briefly the various manners people use to communicate besides speech, explain what augmented reality is, outline shortly what devices we use to put it into practise, and introduce you, again rather briefly, to the relatively unknown auditory analogue to visualization called sonification. The chapter will be concluded by explaining the background of this work, therein summarizing what alignment means in the context of human communication and giving you an idea of how augmented reality can be used for its study.

1.1 Overview

Motivation

Perhaps the most important property that separates humans from all other animals is the way we interact with each other, which first and foremost means our ability to speak. Yet human communication and interaction on the whole is much more multilayered and complex than only spoken language itself. A wide variety of nonverbal means of communication contributes a great deal, and humans usually learn to interpret very subtle and often culturally dependent cues early in life [L. Knapp and A. Hall, 2001].

These natural communication channels have been working well enough for millennia at least, yet new media pose new challenges for these well proven patterns and technical means might be needed to provide solutions to these – in a sense – self-made problems. But what is more, technological progress is reaching a level that makes it possible and even intriguing to think about technical means to improve human communication even when people interact face to face with one another, thereby creating something that could be called *artificial communication channels*.

An emerging technological field called *augmented reality* (or AR in short) seems most promising for realizing this idea. Using head-up displays and other means to directly augment the normal sensory stimuli a person would receive without using any device, it seems perfect for providing the additional input as transparently and unobtrusively as desired.

Current AR research usually focuses on tutoring and assistance scenarios. While some work has been done to explore the use of AR in multi-user scenarios, called *collaborative* or sometimes *shared augmented reality*, there is very little research exploring the use of AR for human interaction itself. Tang et al. [2002] superimposed something like panoramic photographs of one user's surroundings into their equivalent positions of another user's view. While this work is a step in the direction described above, no systematic exploration of using augmented-reality-related techniques for the creation of artificial communication channels has taken place so far. We will go about doing so in Chapters 2 and 5 but first let us have a look at why there might be a chance for current technology to hold its ground alongside the superior human cognitive abilities and under which circumstances it might even complement them effectively.

For remote applications, it is quite easy to conceive how technology can be used to smooth out difficulties which technology itself created. Modern media often cut down on the natural communication channels heavily. For example, interlocutors do not see each other or even if they do, they cannot use the gaze direction either because they do not share the same physical space. AR itself is a technical system that hinders reading facial expressions and gaze in particular. Here, restoring such information with novel techniques is a still quite conceivable yet important goal. Billinghurst and Kato [2002], for example, describe the important role AR might play in enhancing collaborative interaction in the future. In their work, they present a system for remote collaboration. Vertegaal [1999] also describes a system for "mediating joint attention" in videoconferencing applications. Just as we do, they focus on gaze direction. In our setup, though, the interlocutors are sitting at the same table in front of or next to each other, sharing the same interaction space. It is recognized that remote applications may profit from the techniques described in this work as well. However, in the following, we will focus on the creation of new communication channels in addition to those occurring naturally and on the influence these artificial communication channels have on human interaction.

The question remains how today's dumb software cognition and clumsy hardware can achieve something people cannot. In future, computers might gain an advantage over humans by using their access to online information and sensory input which humans do not have the sensory organs to per-

ceive. In order to make these sources of information available in an intuitive way, machines would need some kind of understanding of the data, which they do not yet have. For the time being, there is another advantage personal assistance systems have over human interactants which is that they remain with their user all the time and never have to focus on anything else.

Humans might indeed be better at figuring out what someone is pointing at, but when the dialogue partner is not looking at the pointing person in the right moment, this ability is of no avail. The same applies for the detection of gaze direction. People are great at doing this but they have to look someone into the eyes for accomplishing it. In many situations, though, looking into one another's eyes is cumbersome. This is, for example, the case when sitting or standing side-by-side or otherwise not facing each other or when the attention is bound by performing a common task. In any case, one first has to look someone into the eyes or at his hands before following the gaze or the pointing gesture which may already be distracting and costs time. Also, people sometimes do not take into account the time it takes to follow such a gesture. One person might point "over there" but take his hand down before the other had time to look where he pointed.

Such a collaborative task in which the visual attention is bound to something else is one example where a technical device doing the job might actually be helpful and it is the path we chose to take for our system.

Goals

The work at hand has three main goals. First of all, I will explore some basic prerequisites and fundamentals of possible artificial communication channels in general and will propose a systematization of the artificial channels that are feasible with our current hard- and software. Secondly and most importantly, some of these proposed channels will be implemented and this implementation will be concisely described. The implemented channels will finally be evaluated regarding their usefulness and proposals for further development will be made accordingly.

Before presenting and explaining our system in Chapters 2 and 3, presenting the results we got so far in Chapter 4 and venturing a glimpse into the future in Chapter 5 though, I will give an overview over some of the concepts that are used in the chapters that follow.

Figure 1: Natural nonverbal communication cues cover a wide range, all the way from facial expressions to accessories and clothing. © Photographed by Mo Riza, published under the Creative Commons *by-nc* licence on Flickr.

1.2 Nonverbal Communication

Humans naturally harness a surprising variety of means for communication in addition to words, thereby posing a significant challenge for artificial systems that try to contribute anything to this vast pool of information every human being uses intuitively. L. Knapp and A. Hall [2001] compiled an exhaustive list of these natural nonverbal communication channels of which an aggregation shall be given to clarify the breadth of natural communication cues.

Their taxonomy includes

- body motions and reactions such as
  - hand and head gestures, including touching,
  - movements of other body parts like legs, feet, shoulders or the torso,
  - facial expressions, in particular
  - gaze direction and length,
  - posture and
  - vegetative reactions like pupil dilation, rubescence or perspiration,
- prosodic features such as
  - voice pitch,
  - loudness and
  - tempo,
  - rhythm and syllable length and
  - voice timbre,

- other nonverbal vocal cues as
  - laughing,
  - crying and
  - moaning to name a few and
  - parts of the lexicon usually not considered to be words (like "uh-huh" or "um" in English),
- interpersonal distance (proxemics) and finally even
- the use of objects such as cloths, make-up, accessories and furnishing.

1.3 Augmented Reality

Definition and examples

The term of *augmented reality* was first coined in the literature by Caudell and Mizell [1992] and further explored by Milgram and Kishino [1994], who placed it into a continuum with virtual reality at one end and the everyday world at the other. Basically, it means that the perception of the world people live in and interact with is augmented with artificial stimuli in order to provide additional information or to assist the user in extracting relevant information out of the percepts.

Current research and early applications focus mainly on visual augmentations, but other modalities are included by the definition as well. Azuma [1997] provides a collection of ideas related to AR technology and possible AR applications.

However, applications that are in actual use are currently quite rare because AR still faces many largely unsolved technological obstacles, of which Azuma [1997] and Azuma et al. [2001] provide a good overview. There is some *preliminary* work, though, that is in common use already. One prime example are television broadcasts of sport events, which are often augmented with virtual markings appearing to be embedded into the actual environment. Offside or first down lines or live distance labels next to the athletes, for example, are displayed like this. Yet, there is no interaction with these markings and so it is not AR as it is most commonly understood.

Google Maps might also be considered such a precursor of real AR that is already suitable for everyday use. It definitely can augment real-world data but it does not augment reality itself as the user interacts with a picture, not with what is depicted. From a technical point of view, this is much easier to put into use than real AR as the underlying real-world data do

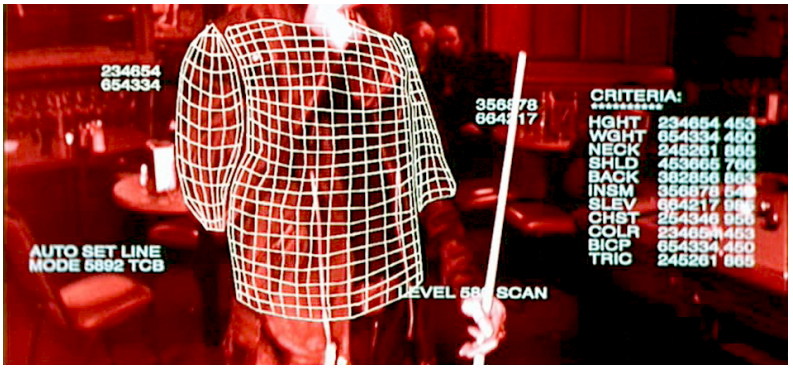


Figure 2: Early science fiction view on augmented reality from the 1991 movie *Terminator 2: Judgment Day*.

not change at a rate that would require a dynamic readjustment of the augmenting data. In any case, Figure 3 demonstrates nicely how neither human-generated information nor unprocessed real-world data alone have the same information density and in many cases not the same usefulness as their combination. Bringing the same level of helpfulness to everyday interaction as well as to special applications is the main objective of current AR research.

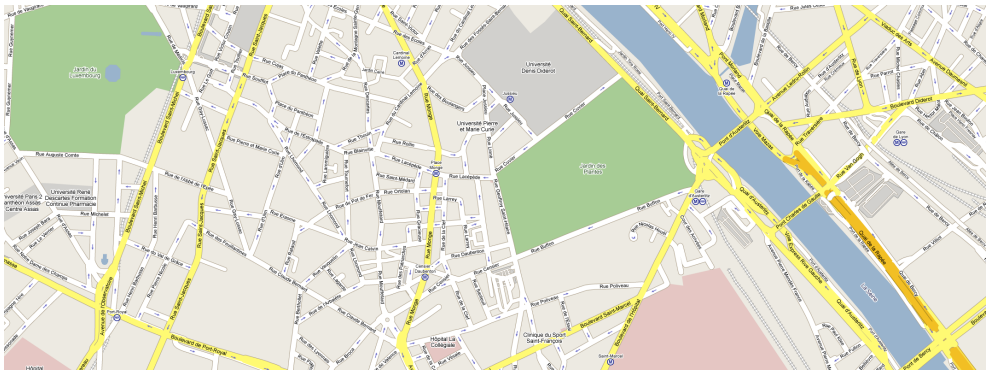
Hardware

The devices most commonly associated with wearable AR are head-mounted displays (HMD).[1] There are two basic types of those, called optical see-through and video see-through. Optical see-through HMDs use a semi-transparent display in front of the eye while video see-throughs use a camera and a conventional display. Both techniques have their specific advantages and limitations [Azuma, 1997].

We decided to use video see-through devices primarily because they allow real objects to be occluded whereas optical see-through generally can only add objects to the visual input (a system developed by Kiyokawa et al. [2000] is a notable exception to this).

To further shield the subjects from their environment and to enable multimodal – namely auditory in addition to the common visual – augmentation,

[1] Although handheld displays might be more common in actual AR applications outside the lab.



(a) A map of the 5th arrondissement of Paris

(b) The same area as an aerial view

(c) A combination of map data and photographic imagery

Figure 3: Google Maps showing the same area as (a) a map, (b) a satellite picture and (c) a hybrid view combining data from (a) and (b)

we use circumaural headphones in our setup. See Section 3.1 for further details on the hardware we used.

1.4 Sonification

As just stated, the augmentations that we add to the sensory input of the users of our system comprise auditory stimuli in addition to the visual ones. While most people have little trouble imagining virtual objects being embedded into a real environment, sound seems to be a less obvious modality. While sound could be used to accompany the visual virtual objects to enhance the immersive qualities of the system, we chose to use it to convey arbitrary data to the user, employing a technique called *sonification*.

Sonification is generally defined to be the use of non-verbal audio to convey information [cf. Kramer et al., 1999]. So just as visualization transforms data into a visual stimulus to make use of the human brain's unsurpassed image processing capabilities, sonification transforms data into sound to use the equally astounding yet different perceptional abilities of the human auditory system.

Hermann [2002] lists some advantages auditory displays have over visual ones. These include

- being *eyes-free*, which means they can be used without demanding visual attention,
- having a high temporal resolution,
- coping better with overlap,
- *backgrounding* is possible, meaning one becomes unaware of a sound right until it changes in any significant way,
- the ability to complement visualizations instead of competing with them like other visualizations would and
- an increased realism when virtual objects produce realistic sounds.

Additionally, the response time to sudden stimuli might decrease because auditory reaction time is generally about 40 ms lower than visual reaction time [Kosinski, 2002] and one might even speculate that using redundant input modalities might further decrease reaction time as the overall stimulus intensity increases. Redundancy, moreover, has the potential to increase the reliability of recognition so that the combination of visual and auditory displays might make the user miss less information compared to the use of only one visual channel.

There are potential problems, too. Hermann [2002] enumerates some, like

- difficulties for listeners to quantify sound properties,
- complex mutual interactions between different sound properties,
- interference with speech,
- the alien appearance of many sonifications as most people are not used to interpret the information that is presented to them in this way, and
- the fact that it is very easy to get annoyed by acoustic input, especially as it is impossible to shut oneself off from it.

The aforementioned redundancy has a drawback, too, as people could suffer from information overload if they failed to effortlessly integrate the information from different modalities.

1.5 Alignment in Communication

Definition

To conclude our introduction, let us take a look at this work's context which lies in linguistic alignment research.

Alignment is a comparatively recent concept in human communication research, first presented by Pickering and Garrod [2004]. It describes the implicit mutual convergence of the internal states of dialogue partners through very basic mechanisms.

On the level of the representation of a certain situation, this means that interlocutors can focus their exchange on the chunks of information that are most responsible for this particular mental representation. They do this by relying on the assumption that their vis-à-vis functionally shares the same underlying structure to represent situation models and thus can be shifted into the same condition by some key stimuli. Obviously, this greatly reduces the amount of information that has to be transmitted, compared to a complete description of all relevant parts of their internal state.

Because the assumption of equal underlying structures usually does not perfectly hold true though, alignment is an interactive process during which the right amount of information that is necessary to reach a common ground is progressively established. It is also an automatic process which means that it goes easy on cognitive resources.

The Collaborative Research Centre (CRC) *Alignment in Communication* (*Sonderforschungsbereich 673*) – from which the project described in this work originates – was founded for the sole purpose of studying this phenomenon. While Pickering and Garrod in their original paper focused on a more linguistic view on alignment, highlighting alignment at the different structural levels of language, the CRC takes a more general approach to alignment as I did with the above characterization. So in the CRC's proposal Rickheit et al. [2005] define alignment very similarly to be

> [. . .] an ensemble of verbal and non-verbal means that serve to increase the similarity in structure of two interacting dynamic systems in a largely automatic and non-reflexive fashion, without an explicit exchange of information on system states.

This definition is not even restricted to human interactants, so in the CRC there are many different research projects, ranging from purely linguistic ones to those situated in the field of cognitive robotics.

Relation to this work

One part of this CRC on alignment is the research project C5 *Alignment in AR-based cooperation*. Its goal is to use augmented reality for the benefit of alignment research in three main ways: (1) To record data from two interlocutors and to analyze this data with respect to alignment that might take place, (2) to interfere with the subjects' interaction by augmenting their perception differently, leading to controlled mis-alignment, (3) to create artificial means by which alignment might be facilitated.

To tackle this last task is what this project set out to do. At the time of this writing, it will not, however, be able to make any progress in alignment research itself because the recording and more so the analyzation part of the project is not finished yet. However, it aims to enable such a progress as soon as these means become available.

2 Preliminary Considerations

In this chapter I will describe what confines artificial communication channels in general, what kinds of artificial communication channels could be realized with the hardware we have at hand and which of these possibilities were actually implemented.

2.1 Technical Possibilities

Basic prerequisites

Let us have a look at the fundamental possibilities and limitations a designer of artificial communication channels is facing in general, without going into the unedifying details of the everyday struggle with image recognition and the like. At the most basic level, any such technical system has to have at its disposal the information it wants to convey to the user, either directly from some means of information retrieval such as sensors or by further processing its input data. This may seem obvious, but it provides a good starting point when thinking about such systems. Therefore one has to contemplate what information exists, how it can be used and how it can be processed to be more helpful.

In a next step, the system has to present the information acquired in one way or another to its user. This should happen in an intuitive and unobtrusive way. If the user is forced to put too much cognitive resources into the interpretation of the presented data, the probability of the system to actually aid the user diminishes. If he, for example, needs to "switch contexts" between the real world and the virtual augmentation or if the superimposed data occludes or obscures important real world input, this obviously has a negative impact on the helpfulness of the augmentation.

Unfortunately, current AR systems can not only easily impair the user's ability to perceive his environment but can also hamper its manipulation.

This is surely about to change since AR systems, like all electronic devices, shrink rapidly. It poses a problem, though, when trying to evaluate the ideas for tomorrow's AR applications with today's equipment.

Our input data

Now what consequences does this very basic picture of AR system design have for the potential capabilities of our own system, whose hardware setup was described briefly in Section 1.3 and will be presented in more detail later on in Section 3.1?

The direct sensor inputs that are available with the current setup comprise

- head position and orientation,
- head movements,
- the user's view and
- his vocal output.

From this we currently[2] derive

- the gaze direction[3] and
- the marker positions from the AR Toolkit (see Section 3.3) software.

From these data we deemed the gaze direction by far the most promising but I will shortly discuss other possibilities further below as well.

Major dimensions of gaze direction display possibilities

We can use the data just listed for the display of one user's area of attention to his interaction partner in a variety of ways. There are two main dimensions by which the different types of common FOV display can be systematized into a general scheme. For a first discrimination, the area of attention can be displayed either discretely or continuously. *Discrete* in this case means, that not the whole field of view (FOV) is displayed as such but a finite number of entities within it are used for the display. These entities

[2] For a list of potential future processing possibilities see 5.2 "Prospects".
[3] disregarding eye movements because of the narrow field of view of the cameras

	discrete	**continuous**
no temporal dynamic/ memoryless	binary highlighting of objects	highlighting of partner's area/volume of attention
temporal dynamic/ e.g. memory	fading highlighting of objects	gradually decaying trace of area/volume of interest

Table 1: Main augmentation possibilities for gaze direction

can, for example, be virtual or real objects. *Continuous* means the opposite of this, which is to say that the FOV is displayed regardless of its current content[4]. Independent from this difference and thus forming the second dimension of the classification, the display could possess a memory of its recent past or it could just show things as they currently are. The variant with memory is true to the paradigm of Section 1.1 which asserts that, with the current state of the art, a system that makes use of its constant presence has the most potential to be of actual help to the user. It can provide information from the recent past the user might have missed. On the other hand, in scenarios where AR is used anyhow, a means to reestablish the data lost by the loss of eye contact through intransparent AR goggles is very important and probably a huge help.

The concept of memory can be extended to any kind of temporal dynamic. Especially a short delay before the highlighting reaches its full intensity might be useful in many cases, to be able to distinguish between a quick glimpse or an accidental glance while sweeping over some objects and an intentional fixation.

Table 1 recapitulates the two axes just described and briefly depicts their intersections.

So combining the attributes *continuous* and *memoryless* yields systems that could display a kind of *vision cone* or a kind of *gaze beam* in 3D or a conic

[4] "Regardless" might actually implicate the wrong thing if understood in its strong sense, as our own continuous visual display is a two-dimensional shape *on the table* which, of course, is an object in the field of view. More sophisticated implementations might even project a continuous two-dimensional field of view on the surface of any object in the surrounding world. This would not change the fact, however, that the content of the field of view is not what such a visualization is about and therefore it is still clearly a continuous one.

Figure 4: Conception of the trace a user's wandering eyes might leave with a continuous 2D implementation with temporal dynamic

section, quadrilateral or other according shape (depending on the assumed or actual shape of the field of vision) on the interaction surface in 2D.

Adding memory to this might manifest as a shape that leaves a trace where it recently passed. In 3D this probably becomes quite muddled very quickly, but for the two-dimensional augmentations, this might be a useful extension. Figure 4 shows a conceptual design of such a system. You can clearly see that the current focus probably lies on the candy in the lower right corner while the eyes wandered roughly from left to right over the table with fixations on most of the objects and only very little visual attention on the empty space in between.

On the other hand, the field of view could also be displayed discretely, which in this case means that only the objects the respective interaction partner can see are highlighted and not the whole area. This idea, too, can appear in a plain and memoryless variety and in one that makes sure objects are not switched on and off instantly but rather their highlight fades in and out according to some envelope function. Figure 5 shows such a so called

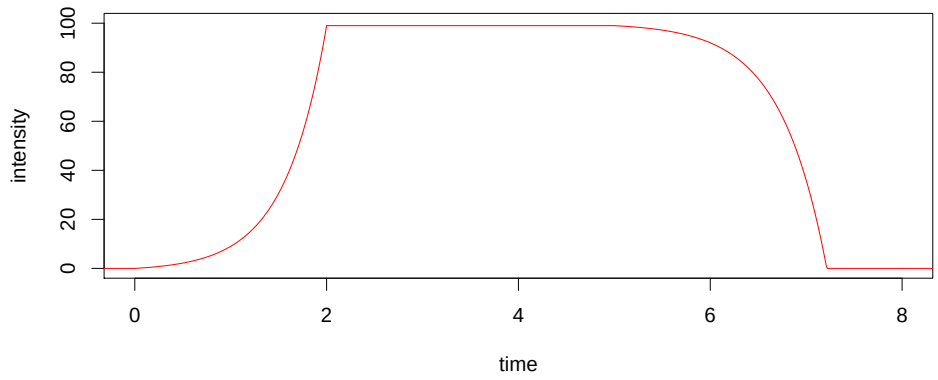


Figure 5: Exemplary attack-sustain-release envelope

ASR envelope where the three phases called attack, sustain and release are nicely visible. While the release phase has the above-mentioned memory function, the attack phase might serve as a low-pass filter to suppress short saccade-like head movements with little or no relevance to the interaction partner[5] or accidental glances at objects that incidentally crossed the line of sight for a very small amount of time during a head movement.

As for how the highlighting is actually done, almost any kind of visual effect filter is conceivable. For this work, a simple colouring was chosen. Real-world objects identified by some object recognition module probably reduce the freedom of choice regarding the highlighting methods compared to the virtual AR Toolkit (ART) objects we use (see Section 3.3). As most object recognition techniques only return bounding boxes, not accurate object surfaces, one would probably resort to rectangles or transparent cuboids around the objects although a fuzzy roundish effect filter might do the job as well.

Another problem for two interlocutors trying to find out each other's field of view is the case when there is no overlap between the two. In this case, the restricted FOV due to the AR devices becomes a real obstacle. The previous VAMPIRE project used small arrows at the edge of the screen to guide the user's attention to some place outside his FOV. However, a study by Hanheide [2006, p. 149] showed that users did not accept this way of guidance

[5]Within the C5 project there is currently a study in the planning phase to find out whether such movements take place and how AR gear affects a person's overall head movements.

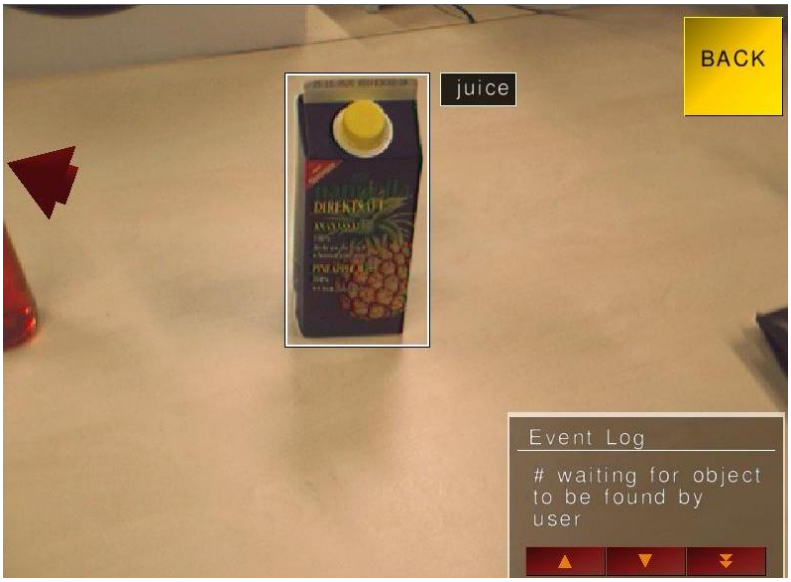


Figure 6: The VAMPIRE system uses arrows to direct the user's attention to some spot outside his FOV. Image used with permission from Marc Hanheide.

very well. Neither accuracy nor helpfulness were rated good. Therefore, we decided to try another technique that currently only works for the 2D continuous FOV display: Instead of a mere perspectively distorted rectangle representing the partner's field of view, the system adds a highlight gradient (Figure 7) to the rectangle's edges, giving it a fuzzy appearance. This means even without overlapping FOVs the interlocutors still see the gradient and can follow it upwards to find each other's centre of focus. It should also be quite intuitive since the same representation is used as for the FOV display itself (that is, a coloured area on the table) and people should be familiar with the concept from the shine of a lamp.

Alternative possibilities of gaze direction augmentation

Outside the scheme presented in Table 1, there are some more possibilities of using the data we have at hand. A technically quite direct way would be a picture-in-picture display of the partner's camera signal inset into one's own view. This idea can be developed further. For example, one could use the position of the other's picture within one's own as an indication as to where he is looking. The work of Tang et al. [2002] I mentioned in Section 1.1 can also be considered such a sophisticated picture-in-picture implementation.

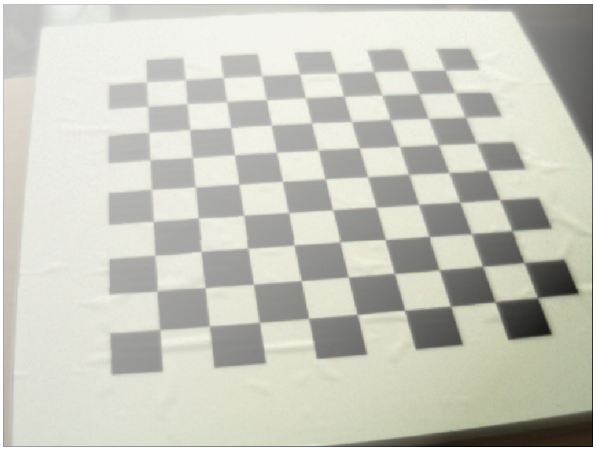


Figure 7: The fuzzy area of the continuous field-of-view visualization. The actual rectangle is beyond the edges of this image, above and left of it.

Finally one could leave the realm of visual augmentation altogether and use techniques from the field of sonification mentioned in Section 1.4. This, of course, opens a whole new range of possibilities which can be grouped in two main categories: (a) Sonifying single events (like "object leaves FOV" or "object enters FOV") and (b) a more or less continuous sonification of the interaction partner's current centre of focus or of the objects currently in view. The advantages of sonifications in general were already presented in Section 1.4. Apart from these general factors, there is another noteworthy difference between the content of the information provided by the auditory displays compared to the visual ones. This is that, while the former mostly provide the same information in a different way, they do so constantly, not only when the user is looking at a certain area. So they could be seen as displaying the interaction partner's FOV in general, not only the *common* field of view of both partners.

Non-gaze-direction displays

As the main input data for our system is the gaze direction, there is not much left to display to the user without further post-processing. There still is the input from the inertial sensor, though, which could be displayed visually (although this would probably be most useful with a downstream head gesture recognition and classification) or a sonification. The latter, though, probably would have a hard time providing information that a SCOTTY[6] soni-

[6] SCOTTY is the name of the 6 DOF head tracker, see Section 3.1 and Section 3.2.

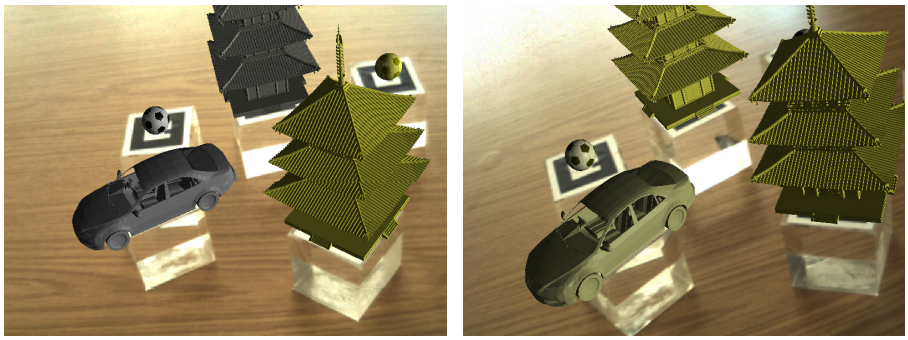


Figure 8: Snapshots of the discrete visual augmentation in action. The left picture shows three unhighlighted and two highlighted virtual objects projected on ART markers. The right picture shows two fully highlighted objects and two which are partially highlighted which means they either just went into the interaction partner's FOV or they recently left it.

fication could not provide. Section 5.2 will look at possibilities further post-processing could enable.

2.2 Implemented Channels

Visual augmentations

In the previous section, some possibilities to visually display the gaze direction were pointed out and systematized using a 2×2 table. However, every cell within this table allows some freedom of how to actually implement the respective concept. Furthermore, in this project, the continuous augmentation with temporal dynamic was not implemented at all. How the other three cells' augmentation principles were implemented will be described in the following.

Figure 8 shows the two discrete systems at work. In the simple memoryless case, objects instantly turn from grey to yellow when they are in the interaction partner's field of view. Adding temporal dynamic means that they change colour, too, but they do so in a gradual way, first fading in until they are fully highlighted, then fading out again as soon as they get out of the partner's view. As described in Section 2.1, different types of envelopes can be used for this and each type can stretch over an arbitrary time frame.

The current default is an exponential onset of 300 ms and a "logarithmoid"[7] decay of 2 s is used. Naturally, the values can easily be adjusted and the defaults should be backed experimentally by future studies.

Auditory augmentations

Independent from the visual augmentations, there can be auditory ones as well. As described in Section 2.1, these can be discrete – that is, event-driven in this case – or continuous as well. The events that trigger sounds in our system are: (a) an object leaves the interaction partner's field of view and (b) an object enters the partner's field of view. The particular sound to play can easily be changed of course; at the moment the sound for appearing objects is a simple 880 Hz 0.6 s percussive sine and the one for disappearing objects is a duller sounding 440 Hz sine with a percussive envelope half as long, mixed with a $1/f$ noise at 30 % the sine's volume.

A bit more thought had to be spent on the continuous sonification which uses the line of sight and turns it into sound. Here, too, a future study would be desirable. It should bring evidence as to which parameters mapped to which sound attributes work best. As no such study has yet been conducted, we took a choice of features and a mapping that we deemed reasonable to begin with. The following table shows which features of the head movements or the line of sight derived from that are mapped to which attributes of the sonification. The mapping is always done using a sigmoid function because for none of the input data there is a definite maximum or minimum value although extreme values should make less of a difference than more common and thus more interesting moderate values. The only exception is the speed of head movement, which cannot be negative. So a function with a horizontal asymptote was chosen which can of course be seen as (and was in fact implemented in terms of) the positive part of an odd sigmoid function.

feature	sound attribute
speed of head movement	amplitude
X-intercept	panning
proximity to own centre of focus	consonance
Z-intercept	fundamental frequency

[7] This is actually the software's terminology for exponential convergence (see Section 3.3).

In a less condensed form this means the following:

1. The pitch is raised when the subject looks up and lowered when he looks down. This might be important for distinguishing easily between an interlocutor who looks down onto the table surface to perform a task and one who looks up to seek conversation.

2. The tone gets more consonant and hence more pleasant when the foci of interest of the two subjects are close to each other and more dissonant the farther they are apart.

3. The stereo position of the focus of interest as the virtual sound source corresponds qualitatively to its position in space. So when the focus of interest of one's partner is to his left, he will hear the sonification more to the left.

4. The more a subject moves his head around, either by changing its position in space or by tilting it, the louder the signal gets. This means especially that there will be no or almost no sound when the gaze direction does not change, which should not only be quite intuitive but also help to reduce the annoyance of the sonification.

3 Implementation

In this chapter we will present what hardware and software components are used to collect the required data, to process them and to feed them back to the user. We will also show how these components talk to each other, and feature some of the more important implementation details.

3.1 Hardware

The system as depicted in Figure 9 uses the hardware components listed in the following. Trivisio GmbH manufactured the AR goggles we used. They customized their *ARvision-3D* model[8] for us to make it use Point Grey *Firefly MV*[9] cameras. These are IEEE 1394 (FireWire) CMOS cameras, delivering an uncompressed 640 × 480 pixels video stream at 60 frames per second. The video signals are then processed by one Lenovo *ThinkPad T61* laptop computer per HMD. These laptops are equipped with Intel *Core 2 Duo* CPUs at 2.2 GHz, 2 GiB of RAM, an nVidia Quadro NVS 140M graphics adapter and an Intel 82801H chipset with integrated audio. As operating system Ubuntu Linux 7.10 is used, running kernel version 2.6.24 with the `CONFIG_PREEMPT_RT` real-time patch set[10] applied. For the user space software that processes the video data refer to Section 3.3. The resulting augmented video stream is then fed back to the HMD via a conventional VGA connector. The Trivisio HMDs feature one 800 × 600 pixel display for each eye; so along with the two cameras this would allow stereo vision. However, we process only a single channel because otherwise the computational requirements would double and a couple of new problems would arise when trying to deliver convincing stereo vision [Drascic and Milgram, 1996]. For our current goals, the benefit of stereo vision was not considered important enough to justify meeting the resulting challenges.

[8] `http://www.trivisio.com/tech_ARvision3DHMD.html`
[9] `http://www.ptgrey.com/products/fireflymv/`
[10] `http://www.kernel.org/pub/linux/kernel/projects/rt/`

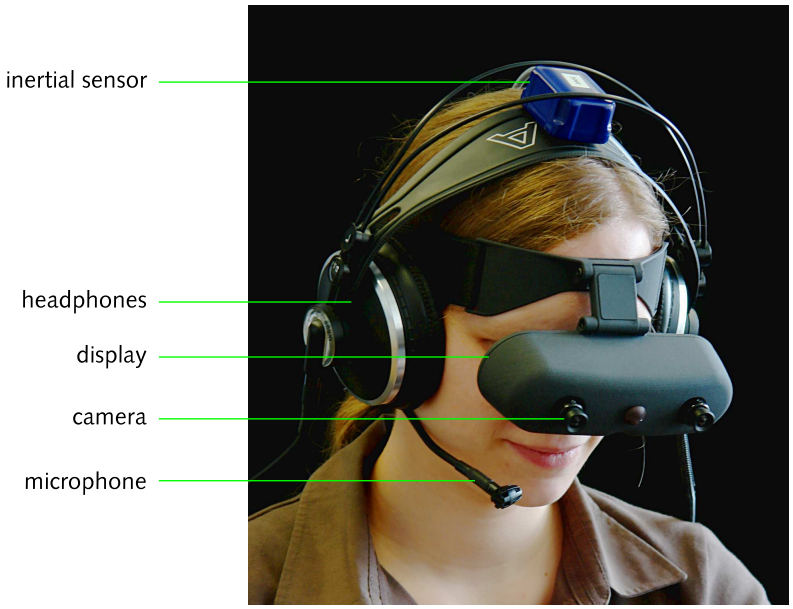


Figure 9: C5 team member Angelika Dierker wearing the complete hardware setup, comprising of a pair of AR goggles, a microphone, headphones and an inertial sensor.

As additional hardware component, an Xsens MT9 inertial sensor can be attached to the head of each user. These 39 mm × 54 mm × 28 mm sensor units contain three solid state accelerometers, three solid state gyroscopes, three thin film magnetometers and a thermometer. The data from these sensors can be used to calculate an orientation with 3 degrees of freedom (DOF) that does not drift (that is, it does not accumulate measurement errors over time). We only use the accelerometers and the gyroscopes though, because in the C5 project we want to record head movements with 6 DOF instead of 3. This has the drawback of being prone to coordinate drift.

The noise root mean square of the accelerometers is 0.7 deg/s and that of the gyroscopes 0.01 m/s^2.

To get a stable head position with 6 DOF, we will use the infrared marker tracking system SCOTTY [Hobein, 2007]. It will use four of the same *Firefly MV*s that are built into the goggles, except that the SCOTTY cameras do not use a Bayer filter for colour images but a 880 nm infrared band-pass filter in front of the lens. The cameras are mounted above the users in such a way that one pair of them always sees one goggle. For tracking the AR gears, passive markers will be attached to the goggles and two cameras per goggle will track its position with the required 6 DOF. The required light of

Figure 10: One of the cameras SCOTTY uses. To the left the LED array can be seen that projects infrared 880 nm light along the cameras optical axis. Right in front of the camera a band-pass filter is mounted with a fitting manufactured by C5 student assistant Alexander Neumann using a 3D printer.

880 nm will be generated by LED arrays next to the cameras. For a picture of one such camera along with the filter and an LED array, please refer to Figure 10.

We do not use any eye-tracking hardware. This draws on the assumption – based on observations from the VAMPIRE project – that there usually are only very few eye movements due to the restricted field of view. This means that, throughout this work, when speaking of gaze direction or the field of view this refers always to the camera's optical axis and the camera's field of view. We assume the person wearing the system to see what the camera sees and we assume the user's eye movements to be negligible. Within the C5 project a survey is being envisaged that is aimed at confirming or disproving these assumptions.

3.2 Framework

Overview

Figure 11 is a diagram of all components needed for the AR setup used in the C5 project (see Section 1.5). The most important parts are the two

LAFORGE systems, which are tasked with grabbing the camera images, augmenting them and displaying them back to the users via their AR goggles, and the central PICARD controlling software. PICARD is the module one would adapt or exchange to implement different tasks or augmentations for the AR users. Both of these central subsystems will be elaborated on in Section 3.3. For a short explanation on the naming scheme, see Box 1.

While the visual output is rendered by LAFORGE, there is also acoustic output to the users which is generated by SuperCollider's[11] `scsynth`, which is basically a digital audio synthesizer. The SuperCollider programming language itself is not needed as PICARD sends network packets triggering prestored parametrized sound units called *synths* directly to the `scsynth` synthesizer. The parameters of these sounds are computed by PICARD and – along with the *synth* names – sent over the network via an application layer protocol[12] called *Open Sound Control* (OSC) [Wright and Freed, 1997].

On the input side of the system, there are mainly SCOTTY and MILES. Both track the head movements of the users with six degrees of freedom, SCOTTY does so using infrared markers fixed to the AR goggles, providing good absolute position and orientation data which do not drift. MILES, on the other hand, uses MT9 inertial sensors fixed to the users' heads, which are suited to observe minute movements. Yet integrating this data to determine the absolute head position would accumulate errors over time. While the MT9 is capable of compensating for this, resulting in a stable and accurate head orientation, the position information is lost in this process, which means translational movements would go unnoticed. SCOTTY is furthermore able to inform the LAFORGE modules about their location in the global coordinate system, which identifies a position relative to some fix point in space (like the middle of the table the users sit at).

The visualization module LAFORGE also feeds data back into the system, namely the positions and orientations of the ART markers. If needed, it will be able to output the video streams from the cameras in the AR goggles.

In future, the overall inter-process communication will be intercepted and stored automatically in the XML format used to exchange data (see next section), without any action needed by the processes themselves. For the time being, all data that needs to be stored gets written on disc in plain text by the processes that produce it.

[11] `http://www.audiosynth.com/`

[12] For an explanation of the ISO/OSI network layer model see `http://en.wikipedia.org/w/index.php?title=OSI_model&oldid=227491223`.

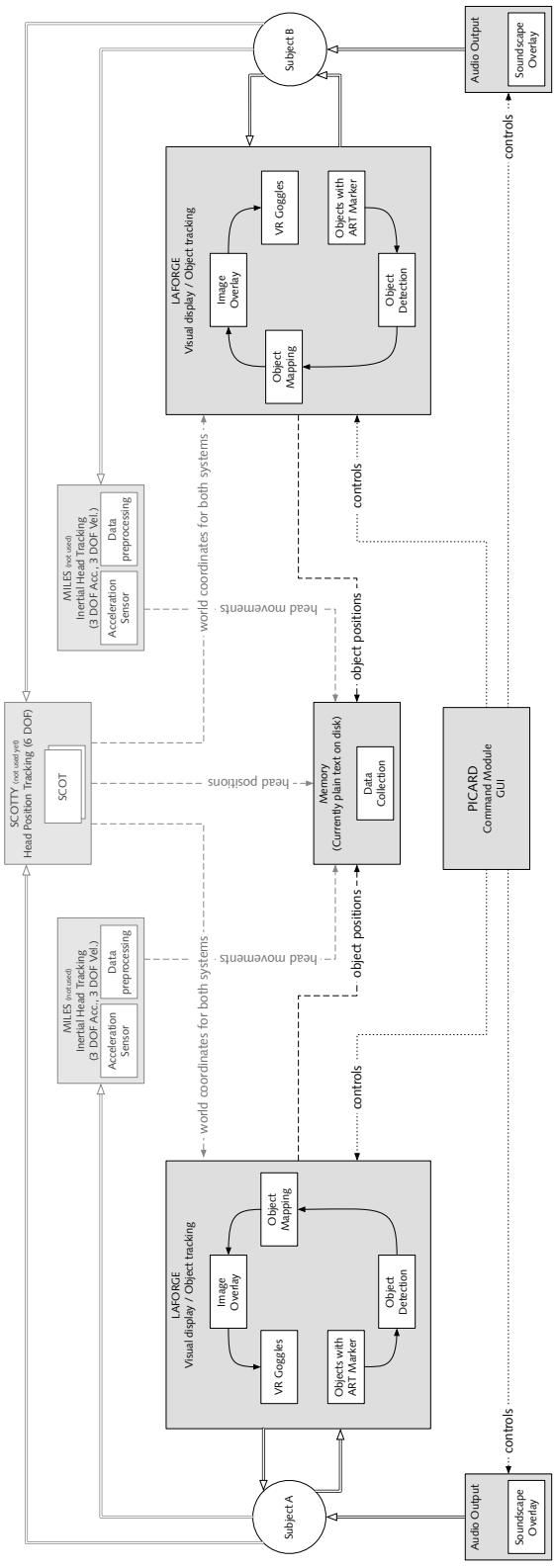


Figure 11: Overview over the modules used for a complete two person setup. The three modules that are greyed out are part of the C5 infrastructure and are relevant to this project, too, but they were not used in the study (see Chapter 4). Dotted lines signify command packets, dashed lines sensor data, and arrows with double lines represent the interface to the user. Derived from a diagram by C5 team member Till Bovermann.

Box 1: The naming scheme
The naming scheme developed because SCOT was the first software module that was to be used. I decided the other modules which were not written at that time had to follow a common scheme and since SCOT reminded me of Scotty (which would even become its name when modified later on to suit our needs) I chose the scheme you see below. The names MILES, LAFORGE, *Taibak*, LEAH, PICARD, SINGH and finally the SCOT-based SCOTTY were invented in this order. Except for *Taibak* they are all backronyms with the following meanings:

MILES: **M**T9 **i**nput **l**everage and **e**ncapsulation **s**ystem

LAFORGE: **l**ightweight **a**ugmented-reality **f**acility with **o**pen **r**eal-world–based **g**raphical **e**nhancement

PICARD: **p**ersonal **i**nterface for **c**ontrolling **AR d**evices

SCOTTY: **S**COT, **c**ustomized to **o**verlook and **t**rack this **t**unnel vision of **y**ours

LEAH: **L**AFORGE **e**xamination and **a**nalysis **h**elper application

SINGH: **si**mple **n**umerated items **g**enerating **h**elper

LEAH and SINGH are only debugging applications and a *Taibak* is not a software module at all but a command package sent by an external application to control what LAFORGE does.[a]

[a] In our case this external application is PICARD, which does not fit the scheme that well but then the acronyms go a long way already.

Inter-process communication

XCF

For all communication between modules the *XML enabled Communication Framework* XCF [Wrede et al., 2004] is used.[13] XCF is a publish/subscribe implementation supporting C++ and Java, designed to use XML messages (and some optional binary attachments). It can be extended to a blackboard system with a software module called *Active Memory*. This work only uses the publish/subscribe functionality though.

This means sensor data as well as visualization commands are encoded as

[13] The only exception being data sent to the audio synthesizer which only understands OSC.

XML and are then sent via TCP to the modules subscribed to the corresponding publisher without the latter having to care about whom it is sending the data to. This alleviated the architecture design and module implementation a lot. The use of XML facilitates future backward compatibility when the use of different software versions with slightly different interface specifications will still work well together.

Anticipated performance issues lead to the design decision not to use the full capabilities of XCF and Active Memory. Namely the XML packets to receive are selected by publisher, not by content, thus saving an extra parsing of the XML content. This introduces some inconveniences, though, especially when a subscriber creation has to be delayed until a certain publisher is available. The worst case example in our system is the case when the creation of a LAFORGE publisher (sending ART marker coordinates) triggers an associated PICARD subscriber creation, which in turn triggers the creation of a PICARD publisher (sending visualization commands), triggering LAFORGE to subscribe to this new publisher while PICARD has to wait for this last subscriber to get ready until it can use its own publisher to send the visualization initialization packets. Active Memory could have made this scenario a lot easier but the anticipated latency increase during runtime was not considered to compensate for the saving of some complexity during the startup phase.

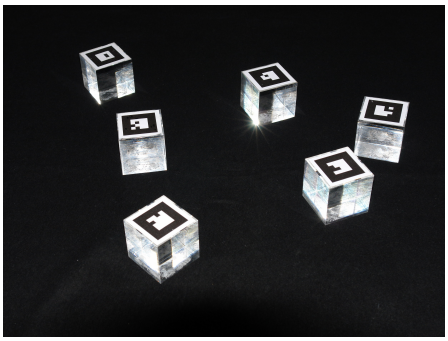
It has to be said though that, in general, the interprocess communication – thanks to XCF – is clean, easy and scalable.

3.3 Subsystem Implementations

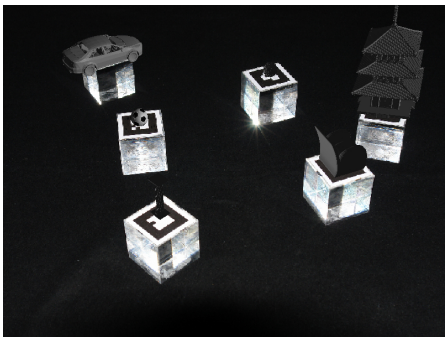
The visual subsystem – LAFORGE

Introduction

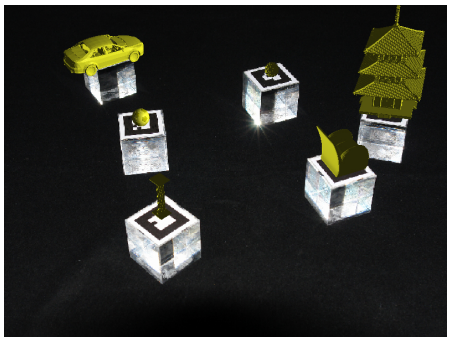
LAFORGE is the module that captures the goggle camera images, augments them according to the previously received XML command packages (called *Taibaks*, see Box 1) and displays them back to the user. It is written in C++ for both performance reasons and its heavy use of the *Image Component Library* ICL [Elbrechter] and the *AR Toolkit* (ART) [Kato and Billinghurst, 1999], a software library to project virtual objects onto fiduciary markers as if they were actually fixed on top of these markers. Figure 12 will demonstrate this process.



(a) Markers without augmentation



(b) Markers with virtual objects



(c) Markers with highlighted virtual objects

Figure 12: Marker cubes without any augmentation (a), with virtual objects using AR Toolkit (b), and with all ART objects highlighted to indicate that they are being looked at (c).

This section only features some selected aspects of LAFORGE's implementation. For more details please refer to the detailed Appendix C.2 "LAFORGE Class Documentation".

XML parsing

The *Taibaks* are parsed using the *XML Template I/O* (xmltio) framework[14] which is an XML parser whose API uses XPath to access single elements. The only exception are the top level tags which are identified using a perfect hashing algorithm by Majewski et al. [1996][15].

[14] `http://xcf.sourceforge.net/docs/xmltio/index.html`

[15] The software that generates the hash function was written by Taj Khattra. Many thanks for writing it and making it freely available. I wrote my own C++ output generator based on

This exception is made mainly for clarity and maintainability reasons because with the hash function a `switch/case` construct can be used instead of a long series of `if/else` statements. In addition, it has the convenient side-effect though of making the run-time complexity of the tag classification $\Theta(1)$ instead of $\Theta(n)$ (with n being the number of possible tag names; assuming constant time for the processing of a single string in both cases) for the string comparisons[16].

As the C++ language does not allow code like **case** *hash*`("SIZE")`: to check for a `SIZE` keyword because this does not count as a constant expression, precalculated constants are used (i.e. **case** *SIZE_HASH*:). These and the hash function itself are created from a list of reserved words[17]. This list can be extended easily and a run of `make`[18] will create a new header file containing the appropriate hash function and the *_HASH* constants.

Continuous field of view display

The continuous field of view display consists of a rectangle that is projected onto the table surface and that describes the exact area the interaction partner is currently seeing with a fuzzy area around it. This fuzzy area should enable the user to intuitively find the current attention focus even when the field of views do not overlap, just by following the colour gradient. Figure 13 shows what rectangle and fuzzy area[19] look like and how they are drawn in OpenGL. How the transformation matrix is calculated that transforms this rectangular shape into the projection on the table will be shown later in this section.

Highlighting envelopes

There are currently three different functions (see Figure 14) that can be used to control the shape of the attack and the decay phase of the highlighting envelope for discrete objects (see Figure 5). LAFORGE has a general framework for this that can be used to change any variable directly or via a setter

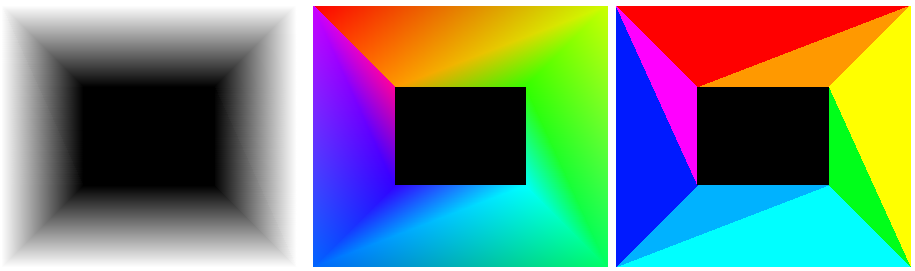
his C code generator but used his parameter generating back-end unalteredly. Source code is available from `http://www.ibiblio.org/pub/Linux/devel/lang/c/mph-1.2.tar.gz`.

[16] A single string comparison is still needed to detect hash collisions from unknown keywords. This does not affect the asymptotic complexity though.

[17] located at `ext/mph-1.2/taibakkeywords`

[18] manually in `ext/mph-1.2`

[19] The rectangle and the fuzzy area surrounding it will in the following simply be called *fuzzy rectangle* as a whole for the sake of conciseness.



(a) Fuzzy rectangle as it looks in the software

(b) Gradual colour change to visualize the order in which the vertices are drawn

(c) Each triangle coloured separately to visualize the patches the fuzzy area consists of

Figure 13: The fuzzy rectangle that is used to display the interaction partner's visual field is implemented in terms of a `GL_TRIANGLE_STRIP`. Figures (b) and (c) illustrate how it is constructed by circumscribing the HSV colour space from red to magenta during its rendering. For the productive version (a) the colour is set uniformly except for the outer vertices whose alpha channel is set to 0. For efficiency reasons the algorithm can be modified slightly to include the inner rectangle itself (not shown).

method as a function of time. When such a *Taibak* is received, a controller object is created that changes a given variable[20] with a constant rate (100 Hz currently). If a new command concerning the same variable is sent while this first controller is still running, it is simply overwritten with the new values. The start value of the new function will be the last value the old controller set. This is surely desirable but it is important to understand the implication that setting a fixed time for any given function does not assure a fixed slope, even if only changing between two fix values, say 0 and 1. This means that if one wants to assure a constant *feel* to an envelope, one has to calculate the internal parameter α oneself. PICARD currently does this but it has to use internal constants of LAFORGE like the controller update rate. It would be desirable to change this in future development of LAFORGE.

One reason for this dependency on internal constants is the fact that the controller currently only uses iterated functions so that it only needs the current value of a variable to compute the next one. To change this to a stateful approach should not raise major problems and is indeed intended in order to implement more complex functions for other tasks than highlighting such as bouncing movements or rotating objects. It was not necessary so far, however, and therefore only iterated functions are used as of now.

[20] Which one this is for any given XML tag has to be hard-coded into LAFORGE though. XML can not be used to manipulate LAFORGE internals directly. This was a conscious design decision.

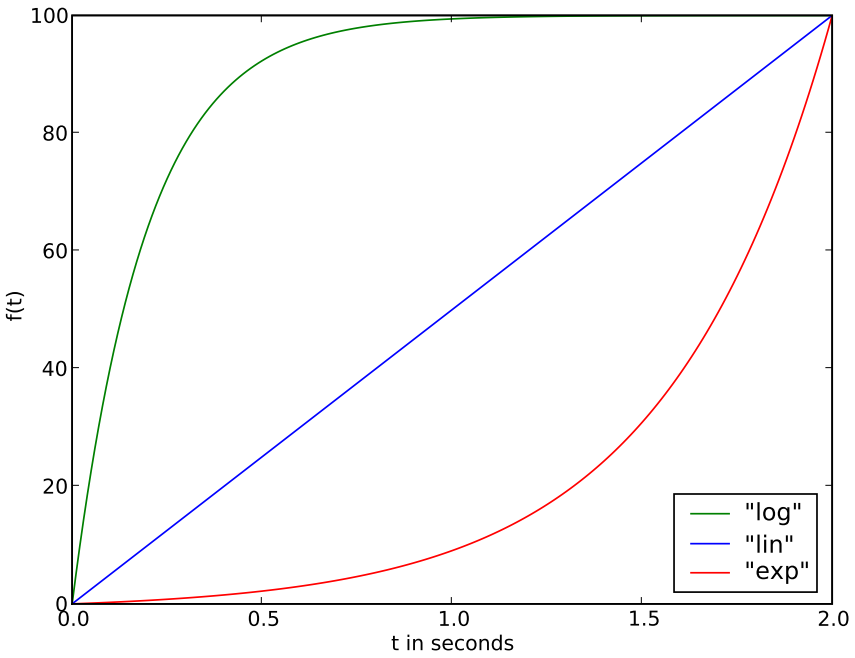


Figure 14: The attack phase of the envelope function $f(t)$ in its three different occurrences described in the text. The parameters $T = 2\,s$ and $\Delta t = 10\,ms$ are taken from the software's default values and $D = 100$ was chosen not to make the function values unnecessarily unsighty. From these follows the parameter a. For the "log" curve this is $a = 0.95$ and for the "exp" curve $a = 101^{1/200} \approx 1.0233$. In the linear case a is obviously $\frac{1}{2}$. The corresponding decay curves are just like the attack curves but rotated by -90°. Please refer to the text for an explanation of the parameters and the tokens in double quotes.

The three functions that are currently implemented are called *linear*, *exponential* and *logarithmoid*. The corresponding values of the XML attribute `<ENVELOPE shape=""/>`[21] have to begin with `lin`, `exp` and `log` respectively. Of these three types, the so called *logarithmoid* one surely requires the most explanation. Actually it does not correspond to a logarithm at all but rather to an exponential convergence. It is not called so, though, because the similar names of exponential convergence and exponential growth (which lies behind the *exponential* keyword) fail to convey their substantial difference (Figure 14). Also, while exponential growth and exponential decay are familiar concepts, exponential convergence – although being very similar to exponential decay – is not. Of course, from a mathematical point of view

[21] This notation actually means the "shape" attribute of the `ENVELOPE` tag.

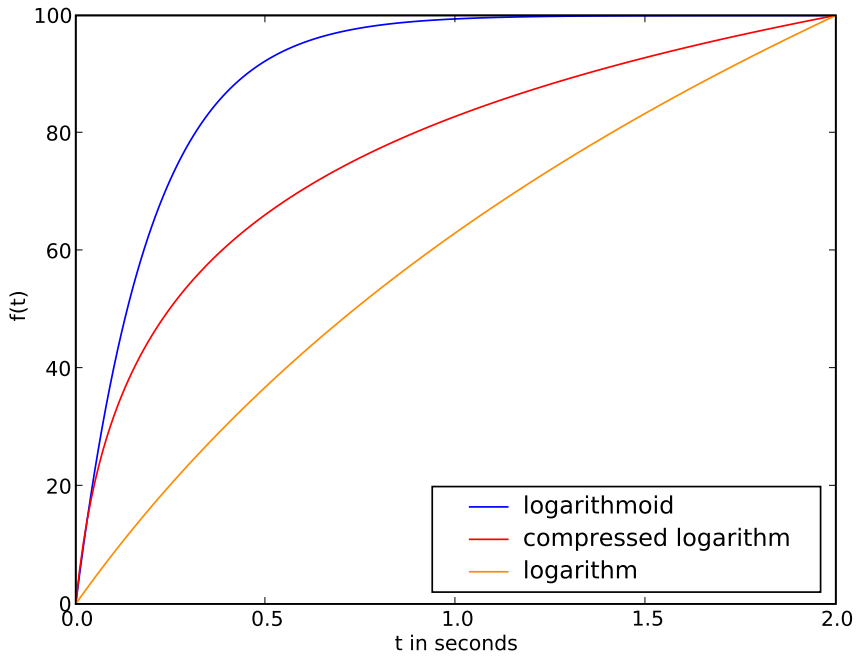


Figure 15: Side-by-side comparison of a real logarithm ($f(t) = \log_b(t+1)$ with $b = \sqrt[100]{3} \approx 1.011$), a horizontally compressed logarithm ($f(t) = \log_b(t\frac{b-1}{2}+1)$ with parameter $b = \sqrt[100]{51} \approx 1.040$) and the caricature LAFORGE uses. The compressed logarithm looks a lot more like the "logarithmoid" curve while the normal logarithm in comparison is hardly distinguishable from a linear graph. The choice of $t \in [0, 2]$ and $f(t) \in [0, 100]$ was made to be consistent with Figure 14.

the logarithm and the exponential convergence have different properties. Perhaps most importantly the latter converges, as the name implies, while the logarithmic function shows proper divergence. From the point of view of a user, though, exponential convergence has a certain logarithmic *feel* to it in that its slope is high at the beginning and then gradually diminishes. Figure 15 shows how one could get the impression that exponential convergence could actually be described as an "idealized" version of the common picture of a logarithm, disburdened from the $x = 0$ singularity and the impractical behaviour for $x < 1$ but still retaining its overall look, especially that of the $x < 1$ part, compressed into the usable range (the middle graph in Figure 15).

The following equations describe how the three functions are calculated. For each function $f(t)$, we will first give its formula, which has a parameter a that corresponds to the XML attribute `<ENVELOPE a=""/>` and controls the du-

ration of the attack or decay. This duration can also be set directly with the attribute `<ENVELOPE t=""/>` which does *not* correspond to the independent variable t but rather to the function's duration T ($t \in [0, T]$). See above for the disadvantage of using this direct way of setting the duration. The second parameter the functions take is the destination value $D = f(T) - f(0)$. They do not, however, take the start and target values as parameters directly. Instead, we will define a second function $F(t)$ which is simply set off from the start value by $f(t)$ so that $F(t) = f(t) + F(0)$ which means that $D = F(T) - F(0)$, too. While the highlighting functions normally go from $F(0) = 0$ to $F(T) = 1$ for the attack and from $F(0) = 1$ to $F(T) = 0$ for the decay, any values are possible here. This is not true for the $f(t)$ which have a specific $f(0)$ and $f(T)$.

We will finally write the function in its iterated form, computing $f\big((n+1)\Delta t\big)$ from $f(n\Delta t)$, with n being the number of iterations of which there are $\frac{1\,s}{\Delta t}$ every second.

The simplest of the three functions is surely the linear one:

$$f(t) = a\frac{t}{\Delta t} \tag{1}$$

From this, these properties follow:

$$f(\Delta t) = a \tag{2a}$$

$$f(0) = 0 \qquad f(T) = D = a\frac{T}{\Delta t} \tag{2b}$$

With Eq. 2b we can calculate a from a given T and D:

$$a = \frac{D\Delta t}{T} \tag{3}$$

And by setting $t = (n + 1)\Delta t$ in Eq. 1, we obtain the iterated equation

$$f(0) = 0 \tag{4a}$$

$$f\big((n + 1)\Delta t\big) = a + f(n\Delta t). \tag{4b}$$

q.e.f.

For the exponential growth, we use

$$f(t) = s a^{\frac{t}{\Delta t}} - s \tag{5a}$$

$$s = \operatorname{sgn}(D), \qquad D \neq 0 \tag{5b}$$

which means that

$$f(0) = 0 \tag{6a}$$

$$\begin{aligned} f(T) &= D \\ &= sa^{\frac{T}{\Delta t}} - s \end{aligned} \tag{6b}$$

$$a = \left(\frac{D+s}{s}\right)^{\frac{\Delta t}{T}} \tag{6c}$$

and whose recursive definition is

$$f(0) = 0 \tag{7a}$$

$$f\big((n+1)\Delta t\big) = a(f(n\Delta t) + s) - s. \tag{7b}$$

q.e.f.

Finally, the *logarithmoid* exponential convergence is an exponential decay which is shifted along the ordinate. Luckily, "shifting along the ordinate" is what $F(t)$ does in relation to $f(t)$ anyway. We write the function as

$$f(t) = -De^{-\frac{t}{\tau}}. \tag{8}$$

The parameter τ is a growth constant for which $f(\tau) = D\frac{1}{e}$. Because $f(t)$ never reaches the target value 0, we define an ϵ so that we can abort the iteration when this value is reached. Regardless of the function used, LAFORGE sets a variable it was instructed to change over time to the designated target value as a last step, right before disposing of the controller. This is done to avert round-off errors. For the exponential convergence this means that it does not harm to return ϵ as its last value, as long as ϵ is close enough to zero (and yes, unorthodox as this may be, it can be negative for $D > 0$). This ϵ can be freely chosen using a constant k[22] which assures that the number of iteration the function needs to reach ϵ does not depend on D:

$$\begin{aligned} \epsilon &= -Dk \\ &= f(T) \\ &= -De^{-\frac{T}{\tau}} \end{aligned} \tag{9}$$

From this we get the parameter τ of Eq. 8.

$$\tau = -\frac{T}{\ln k} \tag{10a}$$

$$\text{which confirms} \quad f(\tau) = D\frac{1}{e} \tag{10b}$$

[22] The software currently uses $k = 2 \cdot 10^{-4}$ which looked good on the plots.

We still need our factor a which, dealing with an exponential function, we introduce like this:

$$f(\Delta t) = af(0) \tag{11a}$$

$$\begin{aligned} a &= \frac{f(\Delta t)}{f(0)} \\ &= \exp\left(\frac{-\Delta t}{\tau}\right) \\ &= \exp\left(\frac{\Delta t \ln k}{T}\right) \end{aligned} \tag{11b}$$

With this we can write f as a discrete system, too:

$$f(0) = -D \tag{12a}$$

$$f\big((n+1)\Delta t\big) = af(n\Delta t) \tag{12b}$$

q.e.f.

The control module – PICARD

Field of view

The conversation partner's field of view is displayed in LAFORGE with the *fuzzy rectangle* described above. The function *FuzzyRect::draw()* generates the shape's coordinates with $(0, 0, 1)$ as its centre, the shape itself being parallel to the xy plane; that is, around the origin, but with $z = 1$. The reason for this offset is going to be explained below. To move each of the n coordinate vectors given in homogeneous coordinates $\mathbf{r}_i \in \mathbb{R}^4$ $(i = 1, \ldots, n)$ of the *fuzzy rectangle* to the place corresponding to the actual field of view to be displayed, LAFORGE expects a transformation matrix $\mathbf{M}$ that is multiplied with each coordinate vector by OpenGL internally according to

$$\mathbf{M}\mathbf{r}_i = \mathbf{r}_i^\star \tag{13a}$$

$$\text{or simply} \quad \mathbf{M}\mathbf{R} = \mathbf{R}^\star, \qquad \mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_n). \tag{13b}$$

So how is $\mathbf{M}$ actually calculated by the *computeTransformationMatrix()* function of PICARD? From SCOTTY it receives three vectors: The position vector $\mathbf{p}$, the direction vector $\mathbf{d}$ and the up vector $\mathbf{u}$ ($\mathbf{p}, \mathbf{d}, \mathbf{u} \in \mathbb{R}^3$).

First we do almost the same steps as the GLUT command `gluLookAt` [Sil, 2006]. This command takes an eye position, a point to look at and an up

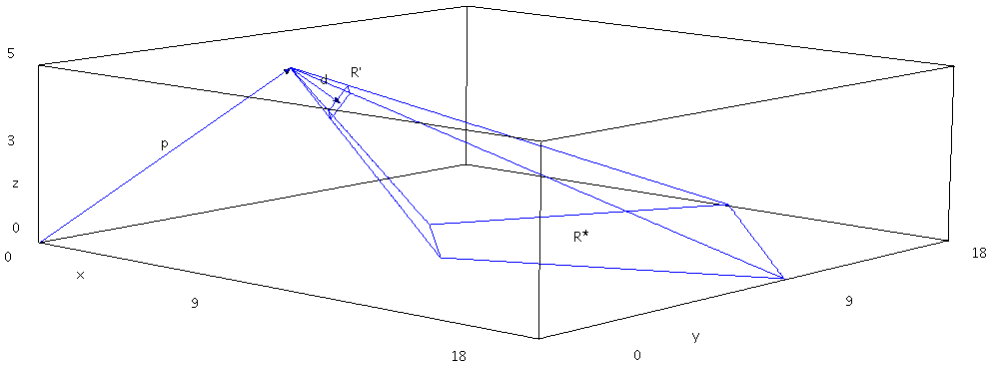


Figure 16: Illustration of the projection of the *fuzzy rectangle* onto the table surface. In this figure, the projected shape $\mathbf{R}'$ is a simple 4:3 rectangle of width 1, not the more complicated shape used in the actual software. Also, its column vectors are not drawn as arrows but rather the shape itself represented by these vectors is shown (the same is true for $\mathbf{R}^\star$). The vectors controlling the projection in this example use the values $\mathbf{p} = (5, 5, 5)^\mathsf{T}$, $\mathbf{d} = (1, 1, -1)^\mathsf{T}$ and $\mathbf{u} = \mathbf{v} = (\frac{1}{2}, \frac{1}{2}, 1)^\mathsf{T}$.

vector. From the two position vectors it calculates a direction vector. Since we already have one we skip this step but otherwise proceed just as the standard library function does: If we cannot be entirely sure that $\mathbf{d}$ and $\mathbf{u}$ are orthogonal, we define $\mathbf{s}$ to be the normalized cross product of $\mathbf{d}$ and $\mathbf{u}$ (so $\mathbf{s}$ is a vector that points to the right of the projection centre's point of view)

$$\mathbf{s}' = \mathbf{d} \times \mathbf{u} \tag{14a}$$

$$\mathbf{s} = \frac{\mathbf{s}'}{\|\mathbf{s}'\|} \tag{14b}$$

and $\mathbf{v}$ to be the normalized cross product of $\mathbf{s}$ and $\mathbf{d}$ (so if $\mathbf{d}$ and $\mathbf{u}$ actually were orthogonal in the first place, $\mathbf{v}$ should be equal to $\mathbf{u}$[23])

$$\mathbf{v}' = \mathbf{s} \times \mathbf{d} \tag{15a}$$

$$\mathbf{v} = \frac{\mathbf{v}'}{\|\mathbf{v}'\|}. \tag{15b}$$

Then we can create a transformation matrix $\mathbf{L}$ in homogeneous coordinates that will transform our coordinate system as to be "looking" along the di-

[23] One could consequently think about checking $\mathbf{d}$ and $\mathbf{u}$ for orthogonality to save us the calculation of one cross product and two normalizations, especially since we get orthogonal vectors anyway as long as SCOTTY does its job right. This is not done yet though and is probably a bit less straightforward than it might seem at first glance due to floating point round-off errors.

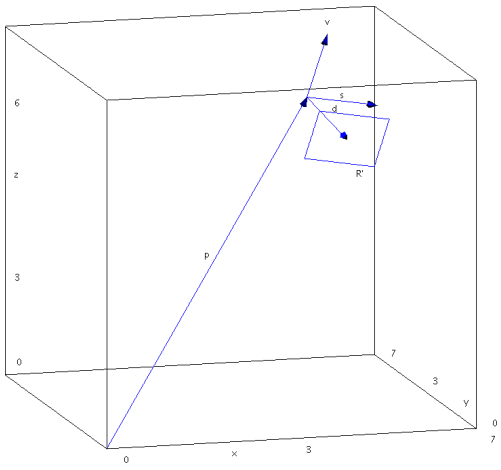


Figure 17: Close-up of the intermediate step before the actual projection. The vectors pointing upwards (**v**) and sideways (**s**) are also shown.

rection vector **d** with **v** pointing upwards perpendicularly:

$$\mathbf{L} = \begin{pmatrix} s_x & s_y & s_z & 0 \\ v_x & v_y & v_z & 0 \\ -d_x & -d_y & -d_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{16}$$

This has then to be translated by **p** with the translation matrix

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{17}$$

Now, thanks to being initially translated by 1 along the z axis, our shape lies, figuratively speaking, like a slide at the tip of the direction vector if this is considered to be looking down at the xy plane from the end of the position vector (see Figure 17). These intermediary coordinates

$$\mathbf{r}'_i = \mathbf{LTr}_i \tag{18}$$

need to be projected back to the xy plane, with **p** being the position vector of the centre of the projection, to yield the final vectors $\mathbf{r}^\star_i = \mathbf{Pr}'_i$.

Aumann and Spitzmüller [1993] describe how to do this using the 4×4 matrix

$$\mathbf{P} = \left(\begin{array}{ccc|c} & \mathbf{A} & & \mathbf{t} \\ \hline -n_x & -n_y & -n_z & \langle \mathbf{n}, \mathbf{p} \rangle \end{array}\right) \tag{19}$$

where $\mathbf{n}$ is the normal vector of the projection plane (so $\mathbf{n} = (0, 0, 1)^T$ in this case) and

$$\mathbf{A} = d\mathbf{I_3} - \mathbf{p}\mathbf{n}^T, \quad d = \|\mathbf{h} - \mathbf{p}\| \tag{20a}$$

$$\mathbf{t} = (\mathbf{n}^T\mathbf{h})\mathbf{p} \tag{20b}$$

with $\mathbf{h} = (p_x, p_y, 0)^T$ and $\mathbf{I_3} = \left(\begin{smallmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{smallmatrix}\right)$.

So our projection matrix is

$$\mathbf{P} = \begin{pmatrix} p_z & 0 & -p_x & 0 \\ 0 & p_z & -p_y & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & p_z \end{pmatrix}. \tag{21}$$

We can now combine all these steps into a single transformation matrix $\mathbf{M}$ that will – applied to each $\mathbf{r}_i$ – yield the vectors $\mathbf{r}_i^\star$ on the xy plane according to the current field of view defined by the vectors $\mathbf{p}$, $\mathbf{d}$ and $\mathbf{u}$. So we can now

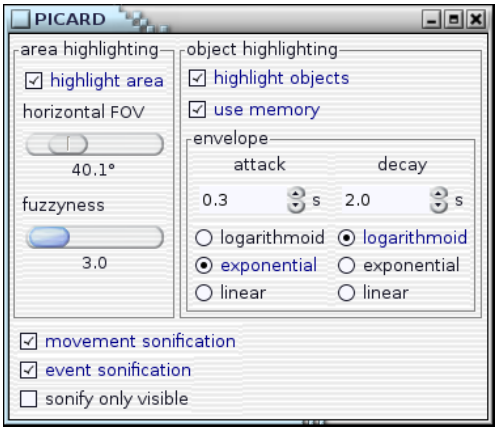


Figure 18: PICARD's optional graphical user interface

write Eq. 13b as

$$
\begin{aligned}
\mathbf{MR} &= \mathbf{PLTR} \\
&= \begin{pmatrix} p_z & 0 & -p_x & 0 \\ 0 & p_z & -p_y & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & p_z \end{pmatrix} \begin{pmatrix} s_x & s_y & s_z & 0 \\ v_x & v_y & v_z & 0 \\ -d_x & -d_y & -d_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{R} \\
&= \begin{pmatrix} p_z s_x + p_x d_x & p_z s_y + p_x d_y & p_z s_z + p_x d_z & \begin{array}{r} p_x p_z s_x + p_x^2 d_x \\ + p_y p_z s_y + p_y p_x d_y \\ + p_z^2 s_z + p_z p_x d_z \end{array} \\ p_z v_x + p_y d_x & p_z v_y + p_y d_y & p_z v_z + p_y d_z & \begin{array}{r} p_x p_z v_x + p_x p_y d_x \\ + p_y p_z v_y + p_y^2 d_y \\ + p_z^2 v_z + p_z p_y d_z \end{array} \\ 0 & 0 & 0 & 0 \\ d_x & d_y & d_z & \begin{array}{r} p_x d_x + p_y d_y \\ + d_z p_z + p_z \end{array} \end{pmatrix} \mathbf{R} \\
&= \mathbf{R}^\star
\end{aligned}
\tag{22}
$$

q.e.f.

The dimensions of the projected rectangle have to be specified by PICARD, too. For them to match the actual field of view of the user, the field of view of the camera model needs to be known. This was already needed for the virtual camera setup in LAFORGE where the FOV α was calculated according

to

$$\alpha = 2 \arctan\left(\frac{s}{2f}\right) \tag{23}$$

with s being the sensor size along one given axis and f being the camera's focal length in the same unit as s (see the *begin_3D_drawing*(**void** $*$) reference).

With this FOV angle α, the size r of the rectangle along this particular axis can be calculated using simple trigonometry by bisecting the isosceles FOV triangle into two right triangles. To visualize this, take the arrows in Figure 17 as segments instead of as vectors. Then $\mathbf{d}$ and a line of length $\frac{r}{2}$ from the tip of the arrow to the edge of the rectangle, parallel to $\mathbf{s}$, form the legs of a right triangle. The leg of length $\|\mathbf{d}\|$ and the hypotenuse of this triangle then include the angle $\frac{\alpha}{2}$. We write down the tangent of this bisected FOV angle in Eq. 24a and substitute α from Eq. 23 in Eq. 24b:

$$\tan\frac{\alpha}{2} = \frac{\frac{1}{2}r}{\|\mathbf{d}\|} \tag{24a}$$

$$\tan\left(\frac{2\arctan(s(2f)^{-1})}{2}\right) = \frac{r}{2} \tag{24b}$$

$$\frac{s}{f} = r \tag{24c}$$

To summarize Eq. 23 and Eq. 24: The width of the "slide" rectangle is $r_x = {}^{s_x}/_{f}$ and its height $r_y = {}^{s_y}/_{f}$ where s_x usually is the horizontal resolution of the sensor, s_y its vertical resolution and f the focal length in pixels.

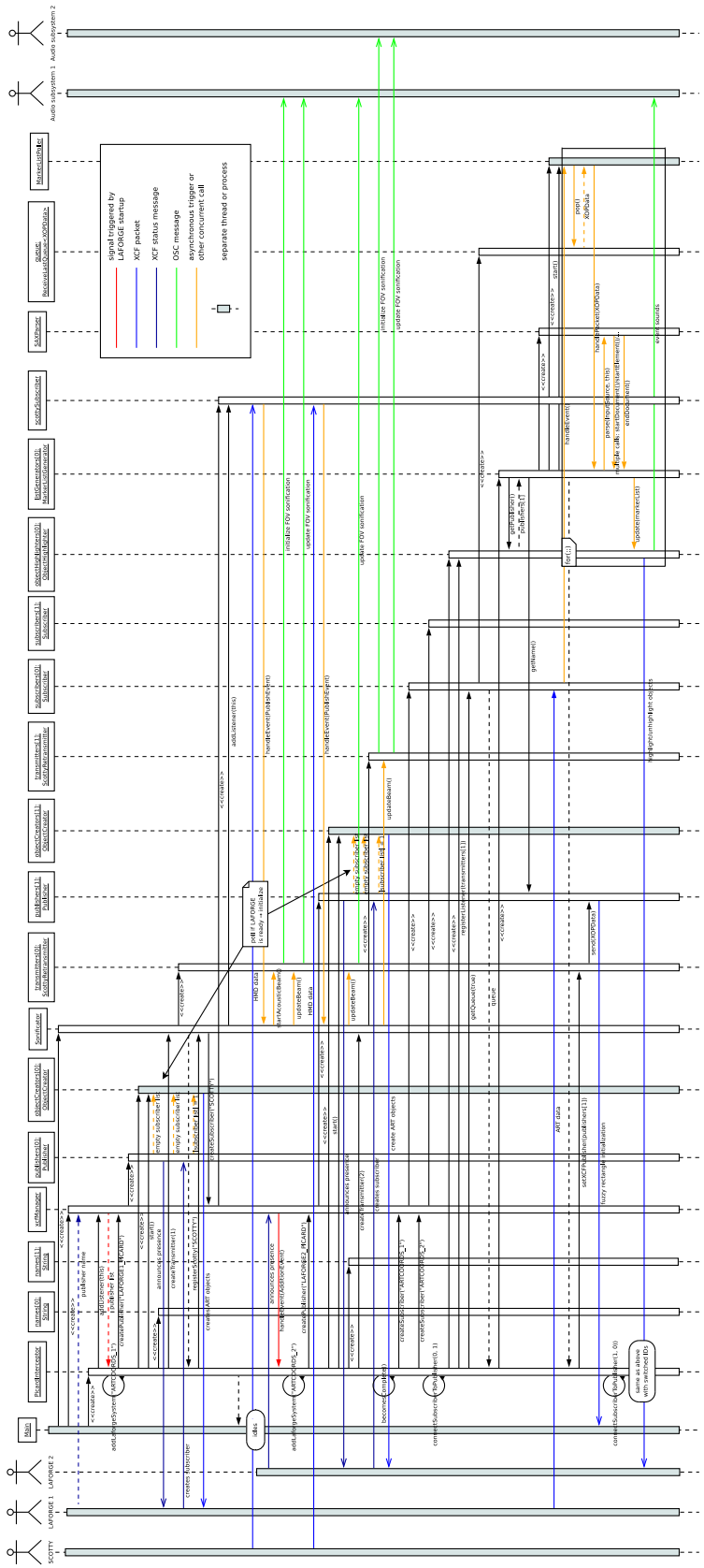


Figure 19: Sequence diagram of PICARD during its quite complex initialization phase. In this example, one LAFORGE system is already running when PICARD is started. The second one appears a short while after. Some arrows that do not contribute anything to the processes or their understanding were left out. This includes many return arrows where the control flow should be obvious. The call of *connectSubscriberToPublisher*`(1,0)` is also missing as well as the associated objects *objectHighlighters*`[1]` and *listGenerators*`[1]` and finally regular XCF input – most notably the SCOTTY HMD data – is not repeated needlessly, not to clutter the diagram more than necessary.

4 Evaluation

In this chapter, it will be explained how we aimed to find out whether the artificial communication channels actually help people, what results these trials produced and how we interpret these data.

4.1 Method

Having implemented the various ways of a mutual gaze direction display described in the previous chapter, we were eager to see how useful they would actually turn out to be. Yet, while the continuous FOV visualization and sonification were implemented, they could not actually be used at the time of this writing due to a lack of suitable head tracking data. So the survey we pursued only deals with the discrete cases – the highlighting of the virtual objects and the sound events when such objects enter or leave the field of view.

We let two collaborating subjects at a time perform a total of two tasks. The subjects were sitting next to each other at a table and were both wearing the AR goggles and headphones. The goggles did not fully shield their eyes laterally, allowing the subjects to see their partner's head movements from the corner of their eyes. During the first task object highlighting and discrete sonification were switched on; the second task was performed twice, once with the two augmentations and once without any. Half of the pairs of subjects started the second task with the augmentations turned off, the other half began with the augmentations switched on and had them turned off later on (cf. Figure 23).

After being explained the mode of operation of the AR system and the visual and auditory augmentations, they had to perform the first task, which was meant to give them a feel for the system in a relatively free collaborative context. The two subjects were first asked to sort six virtual objects[24] before

[24] which were projected onto marker-bearing 5 cm × 5 cm × 5 cm cubes of acrylic glass

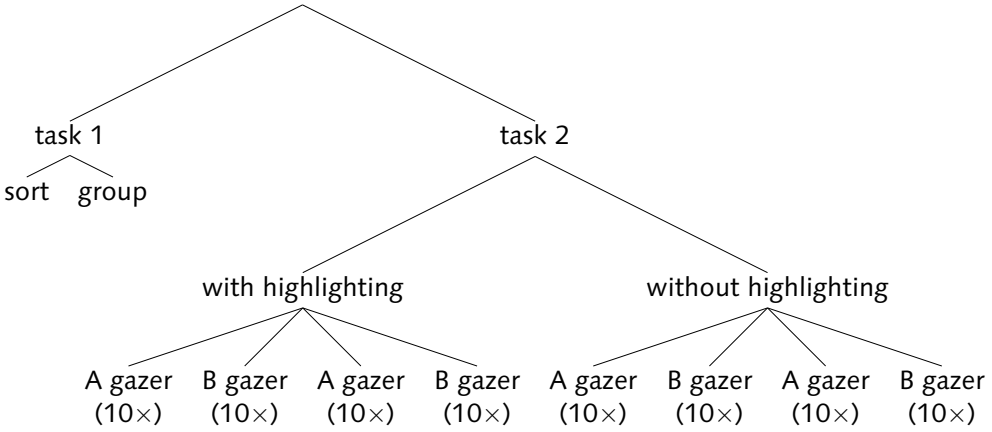


Figure 20: Process flow of the experiment. The tree is traversed left to right, every leaf being an action the subjects had to take. The order of *with highlighting* and *without highlighting* was interchanged for half of the subject pairs. The leaf nodes for the second task indicate whether subject A or subject B takes the role of the gazer and that ten objects are to be chosen before switching roles.

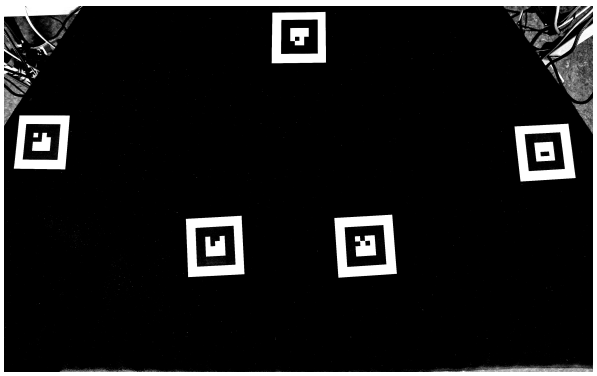


Figure 21: The placement of the ART markers for the second task of the experiment. The subjects were sitting at the side of the trapezoid table that is at the bottom of the image. This side measured 140 cm, the opposite side 70 cm and the depth of the table was 61 cm.

them by size and then to group the objects by any system to their liking. They should solve both subtasks cooperatively and were allowed to speak and act as they pleased.

For the second task, the six movable markers were replaced by five 9 cm × 9 cm cardboard markers that were arranged on the table surface as shown in Figure 21. The subjects were then given one Nintendo Wii Remote controller each as synchronization devices. Now they were assigned two dif-

Figure 22: GUI to control the repetition count and the gazer/searcher role allocation. This application also wrote the times and accompanying status data of the Wii Remote button presses to disc. The controls from top to bottom serve to (1) determine the current gazer and searcher, (2) display the run count (one run being defined to be a single choose–find sequence), (3) toggle the highlighting status in the log file, and (4) switch from introduction mode (displayed as "Run 0") to the actual experiment .

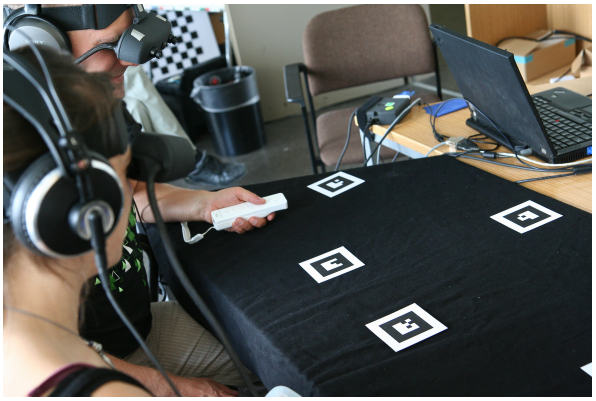


Figure 23: Two interactants solving the second task of the experiment.

ferent roles alternatingly, nicknamed *gazer* and *searcher*. The gazer had to randomly select one of the objects and just focus on it centrically. At the exact moment he picked the object and began staring at it he had to press two buttons simultaneously. This button press made the second Wii controller vibrate for 0.1 s which was the signal for the searcher to look at the same object centredly, too, and press the same two buttons as quickly as he could. The searcher was free to try to follow his coplayer's gaze before the signal. The two players were not allowed to speak or gesture, though, except to verify whether they had actually looked at the same object after the event, which they were actually asked to do.

This cycle was repeated about ten times[25] before switching the roles of gazer and searcher. Every subject incarnated each role twice for each of the two conditions "with augmentations" and "without augmentations".

Before the second task actually started, the above rules were explained to the players and they had the opportunity to familiarize themselves with these rules by trying the game out as many times as they deemed necessary. These familiarization runs were specially marked and taken out of the evaluation.

The times of the button presses were recorded, along with the current role allocation, whether the button press caused a vibration or was extraneous, and if the highlighting was switched on or off. Additionally, the IDs of the detected markers of both LAFORGE systems were saved along with their positions and a timestamp, and the trials were videotaped.

After being finished with the tasks the subjects were asked to fill in the questionnaire whose German original can be found in Appendix D.

4.2 Results

Eight pairs of subjects participated, all previously known to each other to varying degrees, making for a total of $n = 16$ subjects. Among these, 11 were male and 5 female and they were ranging from 26 to 33 years in age.

The first of the multiple-choice questions on the questionnaire was about whether the subjects had any prior experience with AR systems. 9 of them had, 7 had not, but the answer to this question did not predict the outcome of any of the following questions.

The next two questions concerned the uncomfortableness of the system: "How uncomfortable did you find the system of this experiment?" with four answers ranging from "not at all" to "very awkward" followed by an open question "In case you found it uncomfortable: How did this lack of comfort manifest?"

The results for the rest of the multiple-choice questions will be presented as bar charts. For all bar charts, the horizontal axis represents the response

[25] Very quick pairs of subjects led to difficulties of stopping them in time without interfering with the measurements. On the other hand, there were false button presses that misled the run counter. On average, there were 9.8 cycles per turn.

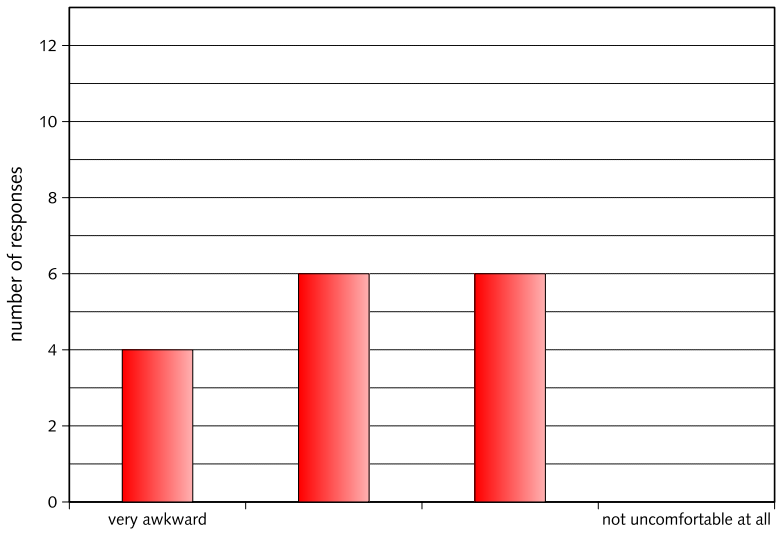


Figure 24: Bar chart of the questionnaire results to the question "How uncomfortable did you find the system?"

options. Unlabelled options were not labelled on the questionnaire either and of course the relative order of the response options is preserved. We chose not to provide a middle option. So all responses had to be either on the negative or on the positive half of the spectrum. For each question at least for one half it will be said what percentage of the subjects chose a response therefrom.

Figure 24 shows that everybody felt somehow impaired by the system. 62.5 % of the subjects chose a response from on the negative (more uncomfortable) half of the four response options.

According to the open question asking for the reasons, people found the goggles irritating, too heavy, hard to adjust, and a strain on head and eyes. The resolution was perceived as too low and the lag as too high. Many complained about the pressure to their heads and one subject even got a headache. Many of the subjects found this issue to be so important that they used the more general questions about the system further below on the questionnaire to express their concerns regarding the uncomfortableness again. Also, when talking during or after the trials, this topic was much raised. What ranked highest among the wishes for a less irksome device were less weight, a sharper image and stereo vision. Beyond the cumbersome hardware, it was the unsteady recognition of the fiduciary markers that annoyed or irritated people the most while using the system, judging by the comments on the questionnaires.

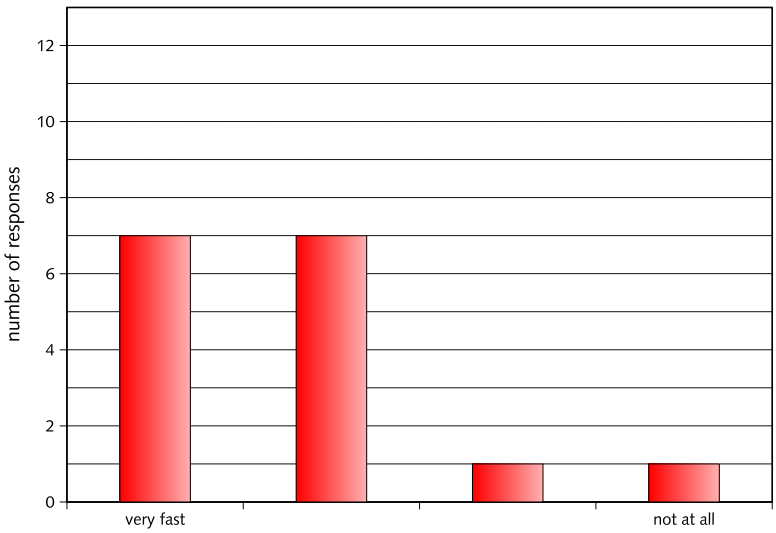


Figure 25: Bar chart of the questionnaire results to the question "How quickly did you adapt to the system?"

The next question prompted for the speed of adaptation to the system (Figure 25). There were four possible answers from "not at all" to "very fast". 87.5 % of the subjects chose an answer from the positive (faster) half. One subject put a remark on the questionnaire pointing out the intuitiveness of the visualization.

The last four multiple-choice questions queried how useful the subjects found the visual and the auditory augmentations and how much they thought they had used them. The scale for the questions about the intensity of usage went in four steps from "not at all" to "very strongly". The scale for the helpfulness questions had four similar options from "not at all" to "very helpful" and an additional fifth option "distracting" that was placed next to the "not at all" option but visually separated from the rest of the scale to indicate the discontinuity.

Figures 26 to 29 show the results from these questions. The percentages of responses from the positive half are: 93.75 % for the visual helpfulness, 100 % for the visual usage, 0 % for the auditory helpfulness, and 0 % for the auditory usage. Nobody found any of the augmentations distracting.

We did not find any significant pattern in the time measurements. Figure 30 shows the means of these measurements per subject pair, broken into runs with highlighting and those without. The box plots of the measured search speed data in Figure 31 further illustrates the large variance of search times

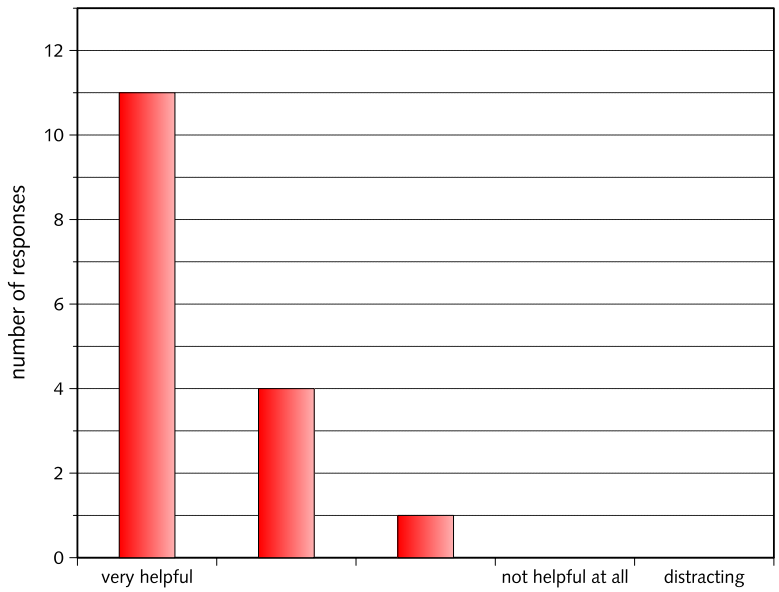


Figure 26: Bar chart of the questionnaire results to the question "How helpful did you find the visual highlighting?"

within and between the individual pairs of subjects and the lack of any apparent structure. The breakdown of the boxes is the same as for the columns in Figure 30: per subject pair and per state of the highlighting. Outliers with more than two standard deviations difference from the mean were removed.

From the video the error rates were obtained by evaluating the oral verifications. Figure 32 shows these error rates as percentages. Except for the three pairs whose columns are rightmost in the figure, few errors were made in any condition. Said columns belong to subject pairs who decided not to look each other into the eyes at all, which increased their mean error rate more than sixfold.

4.3 Discussion

The resonance to the visualization was very positive, judging from both the multiple-choice answers and those to the open questions. This approving view of the visualization was further confirmed by spontaneous oral comments during or after the experiment[26]. This positive personal assessment was not, however, reflected by the performance in a significant way.

[26]like after switching off the highlighting: "This is no fun [like this]." or even "And how are we supposed to solve this [without the highlighting]?"

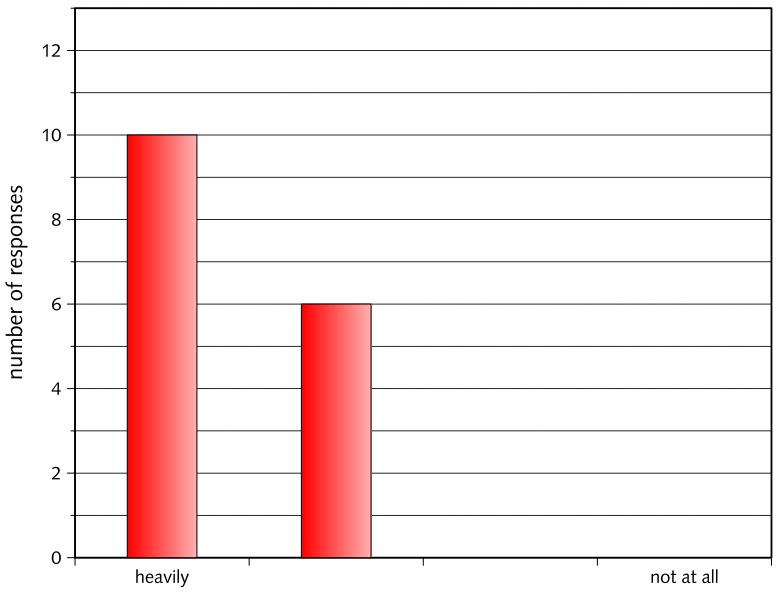


Figure 27: Bar chart of the questionnaire results to the question "How much did you use the visual highlighting?"

This may have several reasons. For one thing, the relatively few objects were positioned far apart from each other. This was done to partially compensate for the current lack of a possibility to distinguish between objects that the interaction partner was directly looking at and those that were only detected at the edge of the camera's field of vision. Judging by utterances during the task this peripheral object detection often went unnoticed even by the person having this object in his field of view[27]. This relatively large amount of space between the objects made it quite easy for many subjects to tell which object their partner was looking at just by seeing the head movements at the corner of their eyes. One pair of subjects was even able to look at their respective partner all the time and tell which object he was looking at by heart, without ever looking back onto the table. A randomization of the objects would have prevented this last way of "cheating" in a way, which would have made sense since there are few real-world situations in which such a strategy could actually be used. But the more general lesson from this probably is that the implementation of a "laterality display"[28] and

[27]There were remarks like "The pagoda is yellow, too. Why is that?" – "I have no idea.", "There were two yellow ones, strange.", and "I *thought* to be looking at the sofa."

[28]Which may for example just result in the highlighting to have a different colour depending on whether the object is near the centre of the partner's FOV or towards its edge. This is a feature that has been in the pipeline for quite a while and so will probably be implemented soon.

a more realistic scenario with more objects closer to each other might shift the actual usefulness into the direction of the already perceived utility.

Some subjects actually proposed the implementation of a kind of a line of sight display or arrows to direct the user's attention towards his partner's. So the employment of the already written head-tracking visualization will meet both these perceived problems and is hence a most promising approach.

Noteworthy but hardly surprising is the distinctive drop of accuracy for the three rightmost columns in Figure 32(b): These were the cases in which the subjects decided only to use the corner of their eye to determine where their partner was looking instead of direct eye contact. Although the error rates are also slightly higher for the rest of the pairs when having their highlighting turned off, this can be explained by chance and a small irritation caused by the highlighting deactivation. Except for the three cases mentioned, the overall error rates were very close to zero anyway, so single mistakes already had a large effect.

The wearing comfort was not only given bad marks but it was also by many considered the major reason not to use the system outside the lab. As stated in Chapter 1, this is not much of a surprise. All we could hope for with this study was a proof of the concept and an idea of its usefulness once the fundamental issues of object detection and cumbersome hardware will somehow be resolved. Considering these conditions, the results are more pronounced than they could have been.

Some subjects noted that the highlighting was hard to see under certain circumstances (like for certain objects – cf. Figure 12(c) –, when looking from certain angles or with badly adjusted goggles) and proposed highlighting in red instead of yellow or rectangles around the objects. An adjustment of the OpenGL material and lighting parameters would probably also fix this problem, but then more colourful objects might worsen it again.

The auditory augmentation on the other hand was not rated nearly as well as the visual one. Neither was the auditory presentation of data perceived as very helpful nor did the subjects state to have used it a lot. First of all, though, it has to be argued that auditory stimuli, mostly unlike visual ones, can be *backgrounded*, which means that they are still processed but on a subconscious level. It is hence possible that the subjects underestimated their own usage of the auditory cues. However, since the performance was not influenced significantly even by both sound and visual highlighting at the same time, the low usefulness estimates seem correct. It could even be said that the subjects overestimated the actual usefulness of the visual augmentation in a way. In any case, as the amount of information encoded in

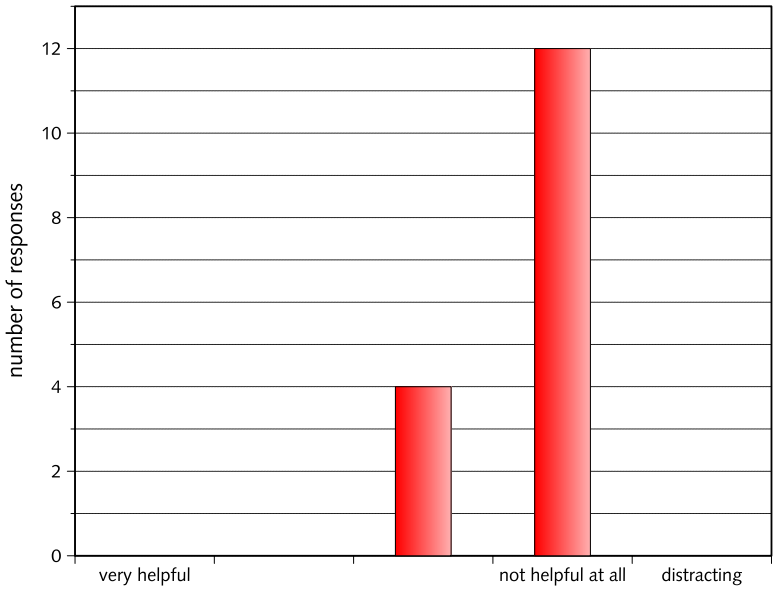


Figure 28: Bar chart of the questionnaire results to the question "How helpful did you find the auditory display?"

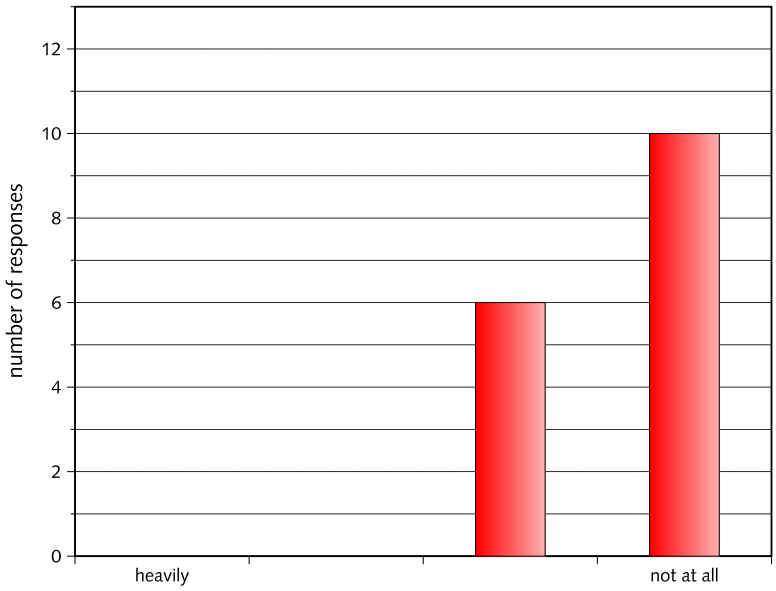


Figure 29: Bar chart of the questionnaire results to the question "How much did you use the auditory display?"

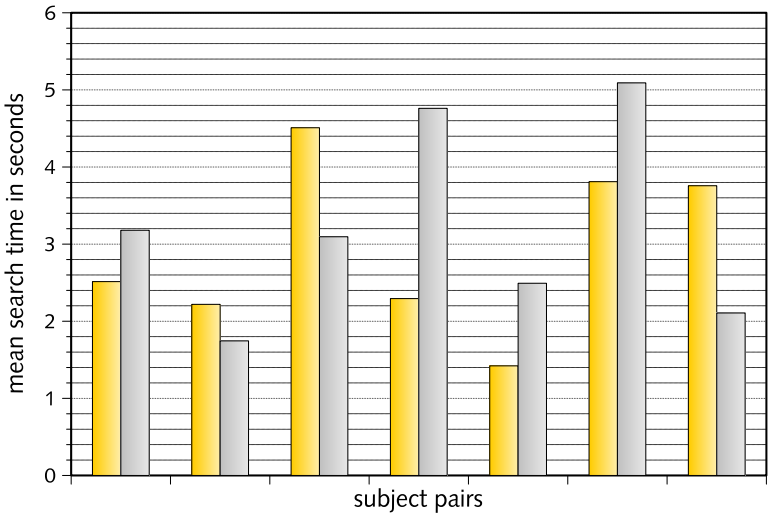


Figure 30: Comparison of mean search times with highlighting (yellow columns) and without it (grey columns). One data record was corrupted so there was only data from seven subject pairs.

the auditory augmentation is less than in the visual one, its general shortfall is not overly surprising, but the extent of this contrast is unexpected. Some participants said they had used the auditory cues as a kind of activity monitor; this is probably the most one could expect from this simple kind of sonification. Even for this use it was compromised, though, by the fact that most subjects tended to move their head very fast to a certain spot when choosing a new object. Fast movement leads to a blurred image which in turn impairs the recognition of the ART markers. So there were no sounds played although the line of sight crossed several objects.

Some subjects proposed to make the presented sound depend either on the objects seen (which would make it the auditory analogue of the current discrete visualization, except that it would not be restricted to the own field of view) with each object having its own specific sound, or to encode things such as the direction of the partner's focal point into a stereo signal. This is encouraging as the current continuous sonification already includes these latter data as well as some other.

The previously mentioned blurring and subsequent temporary deactivation of the virtual objects also diminished the usefulness of the visualization. Colour is a very salient and very effective feature to draw people's attention to one object out of many [Wolfe and Horowitz, 2004]. So, in principle, one quick glance over all of the objects on the table would reveal the ones that

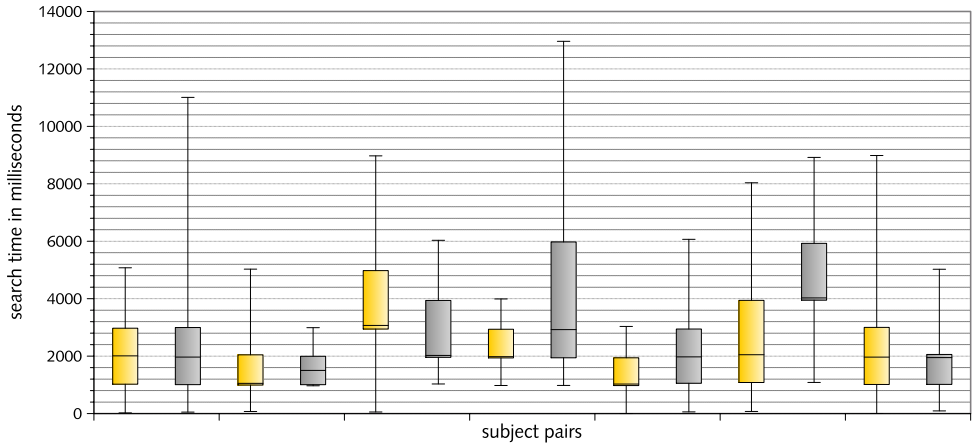
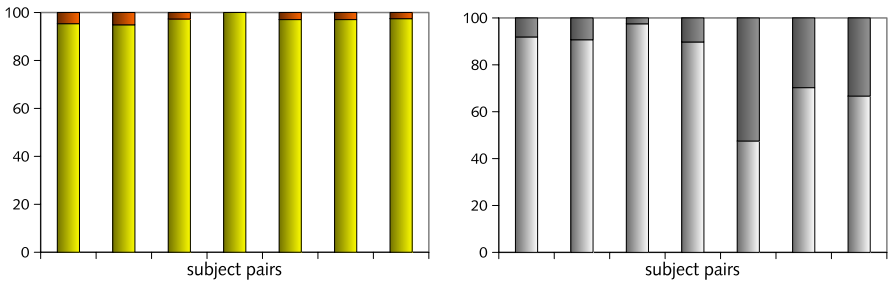


Figure 31: Box plot of the search speed. Yellow boxes represent activated highlighting, grey ones deactivated. The pairs are in chronological order. Outliers with more than two standard deviations difference from the mean were removed.



(a) Error rates with highlighting (b) Error rates without highlighting

Figure 32: Error rates in the object-choice task with highlighting (a) and without (b). The upper part of each column is the percentage of wrong guesses, the lower part is the percentage of right guesses. Only the first guess per object chosen by the gazer was counted. The three rightmost columns in each of the two figures are the subject pairs who chose not to look each other into the eyes, not even when being deprived of the highlighting.

are highlighted. In fact, though, one has to look at one marker at a time and fixate it for a short while to allow the image to stabilize and the virtual object to appear. This is far less effective than it could be but little can be done about it at the moment.

All in all, the results of this first study are almost surprisingly positive, not only in the face of the remaining technical obstacles, but also considering the small amount of information that was conveyed compared to the techniques using head tracking that are waiting to be tested and furthermore considering the challenging direct competition with the largely intact natural channels.

5 Conclusion

In this final chapter I will recapitulate the content of the previous chapters, and hazard a guess how this work could be continued and whether it will change people's lives.

5.1 Summary

Humans naturally use a variety of communication channels to a great effect. Nonetheless, technology has reached a level that makes it interesting to think about complementing these natural channels with artificial ones. Augmented reality provides the means to do that.

One type of artificial communication channel conveys to the user what an interaction partner currently looks at or where he looked a short while ago. This can be done by displaying the gaze direction directly or on the basis of the objects that are being seen. Also, this data can be visualized as well as sonified.

We implemented four distinct modes of data presentation, which can be used together or separately:

- A highlighting of virtual objects by changing their colour when they are in the other's view, using a freely configurable temporal envelope,
- an outline of the interaction partner's field of view on the surface of a table, using an optional surrounding colour gradient to intuitively guide the user's gaze towards his partner's,
- an event-based sonification of objects leaving and entering the partner's view and
- a continuous sonification of the horizontal position of the centre of focus, its height, its proximity to one's own centre of focus and the speed of the partner's visual movement.

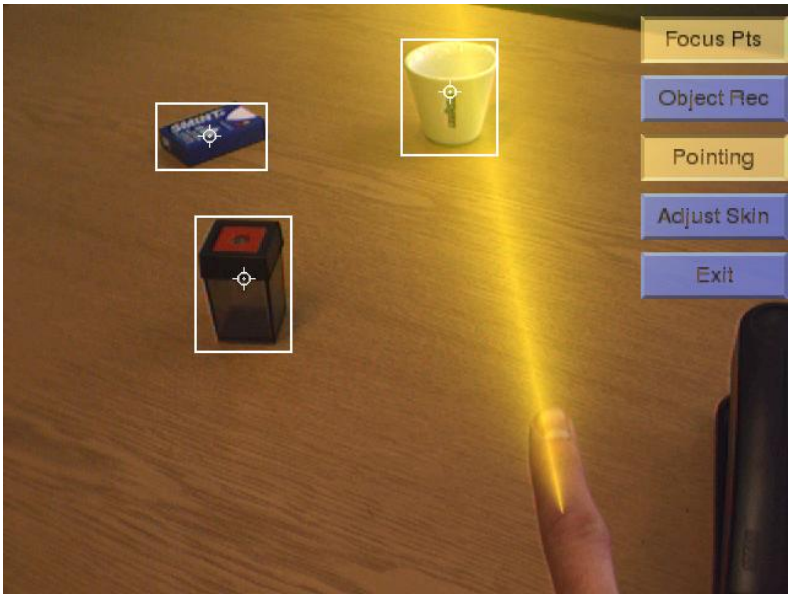


Figure 33: Hand gesture visualization from VAMPIRE [Heidemann et al., 2004]. Image used with permission from VAMPIRE team members Holger Bekel and Ingo Bax.

We tested the highlighting and the event sonification using an object-choice task and found that the subjects accepted the new technique surprisingly well, given the small amount of information that is encoded in these two channels – compared to the more sophisticated head-tracking–driven augmentations. The overwhelming majority found the visualization helpful or very helpful, while the simple sonification was not perceived to be helpful. There is no performance gain yet, but we have reasons to be optimistic towards the two other already implemented augmentations and further development.

5.2 Prospects

Future communication channels

From the raw input data listed in Section 2.1, one can obviously extract much more data than we do currently. This additional information might for example include

- voice pitch,
- distinct head gestures,

- uttered keywords,
- hand gestures,
- laughing [Truong and Leeuwen, 2005] or similar vocal characterizers [Zeng et al., 2007] or
- real objects in the field of view.

Hand gestures are probably the most promising of these choices and would be a great addition to the existing FOV displays. Figure 33 shows how such a pointing gesture visualization looked in the VAMPIRE project and could serve as a model for an implementation in our system. The implementation shown in this image counts as a 3D continuous visualization, but the general classification from Section 2.1 could surely be applied here, leading to discrete visual displays as well as audible ones.

Voice pitch, spoken words and vocal characterizers such as laughing are less likely to provide much aid to most people when displayed visually. They are highly salient audible features and are unlikely to be missed by an interlocutor in a dialogue and even in small groups. On the other hand, they might be all the more helpful for hearing impaired persons. In situations with more than two interactants or when being distracted, for example by a common task, an unobtrusive visual display of these important cues could provide a great help. A simple speech detection display, perhaps showing the direction the voice is coming from, might help hearing impaired persons who can lipread. A voice pitch graph (cf. the more detailed but also more flamboyant voice pitch *spectrogram* in Figure 34) provides more information and even deaf persons who have no way of understanding spoken language might learn to interpret basic intonation patterns. Figure 35 illustrates that one of the most important vocal nonverbal cues – laughter – looks quite striking in a spectrogram and might therefore be detected automatically [Truong and Leeuwen, 2005] to display it with an unostentatious icon.

As with the rest of the artificial communication channels, whether these techniques actually provide any real help fully depends on the hardware used. Few people will want to wear heavy gear that makes them look strange just to get some nice helpful audio cues visualized.

I just presented some ideas for new data sources that could be exploited for artificial communication channels. But even the possibilities of gaze direction displays were not exhausted implementation-wise. There is no vision cone yet, and since the *fuzzy rectangle* already is implemented using OpenGL, it would only be required to add the eye point as a vertex to every corner. This is too simple an extension not to try it out. On the sonification side, a continuous sonification of the objects seen is missing. One implementation of this concept is a soundscape that is created from the ART

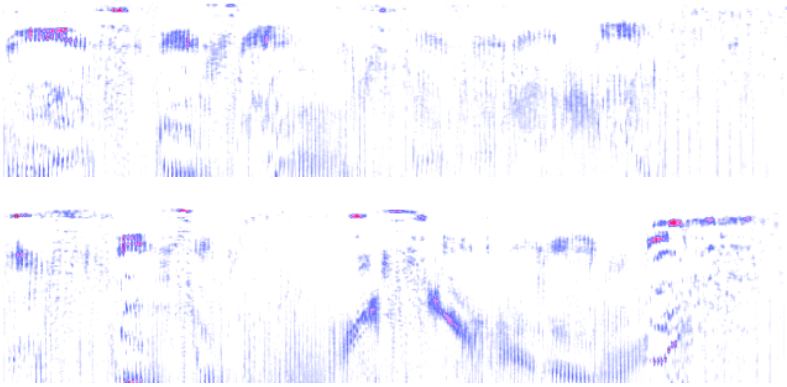


Figure 34: Voice pitch of the same phrase with different intonations

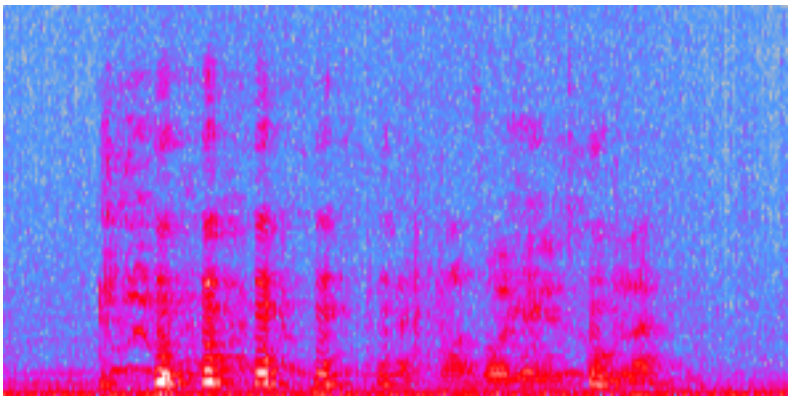


Figure 35: Spectrogram of laughter giving way to speech

objects currently in the other's view. This soundscape could be generic and thus generalizable or each object could have a sound with which it is intuitively associated (like a bouncing sound for a ball). Picture-in-picture is a visualization technique we do not use yet, but it would be interesting to see how it performs compared to the other display possibilities.

Finally, the trials revealed one more useful special case of gaze direction display: This is a notification (visual or auditory) when someone is looking at you. Due to the restricted field of vision it was very difficult to get a subjects' attention when they were distracted, for example by looking at the table. Very often they would not notice being talked to if not called by their name. While for the only two users of the current system this is less of a problem, in general this impairs human communication a lot and could easily be helped with a slight extension of the present means.

Mobility

It is difficult to say, whether, in a scenario in which both interactants are sitting at a table, the system is more helpful when both are sitting next to each other where they can see each other's head movements from the corner of their eyes, or when they are sitting in front of each other where it is not so cumbersome to look up to see what the partner is looking at. To investigate these two cases in a study might therefore be quite interesting.

It is not hard to imagine, though, that leaving the table and moving about would already drastically improve the helpfulness of the system. It becomes much easier to lose track of your interaction partner's focus of attention if you have some business of your own and there are few sensory cues to help you once the other is no longer right in front of you. On the other hand, this would result in much higher demands for the system in terms of mobility, range and accuracy.

Multiple users

The system is not in principle limited to two interactants. It is obvious that under normal circumstances you can keep a close eye on one of the people around you at most. So, say, differently coloured highlights with a long decay time should have much potential here. On the other hand this bears the risk of overburdening the users who would have to remember the assignment of colours to persons if there is no effective AR cue to take this extra task from him.

More studies

It is obvious that the study presented in Chapter 4 only covers small parts of what should be investigated. Some parts that are already implemented were not included in this study and, moreover, it became evident that some minor improvements – such as a feedback about the proximity to the centre of the field of view – would probably increase the helpfulness a lot. In addition, even for the channels used in the study presented above, more scrutiny is needed to differentiate between the effects of the various artificial communication channels, be they visual or auditory. Finally, each of these channels has parameters that were chosen randomly so far.

There are the following variables for the already implemented techniques alone:

- attack envelope shape
- attack duration (with notable special case $T = 0$)
- decay envelope shape
- decay duration (with notable special case $T = 0$)
- highlight colour/intensity
- choice of event sounds
- parameter-attribute choice and mapping of the sonification
- allow or disallow gestures
- allow or disallow speech
- positioning of the interactants at the table

Finding adequate tasks and scenarios will be a further crucial challenge. After all, it is to be expected that the various gaze direction display types are suited for different applications and conditions and even their respective parameters might be tailored to perfectly fit a particular setting. To find general rules that provide your ready-made artificial communication channel for any given application is probably a long-term objective.

Extending the range of scenarios is interlinked with other issues previously mentioned such as mobility and multi-user capability, and it might suggest further improvements such as an object-recognition module to deal with real-world objects.

Communication research

As the idea for this work was born within the C5 project from the wish to better understand human communication, this is a path that shall be further explored. The idea of gaining insight into human communication, especially regarding alignment, by recording and selectively impairing or enhancing it remains intriguing. Other research might investigate the impressive human capability to quickly adopt to such novel forms of input and to effectively integrate these forms with the variety of channels they already use.

Virtual environments

The scheme of Chapter 2 and the idea of enhancing human communication using artificial communication channels might not only be used for aug-

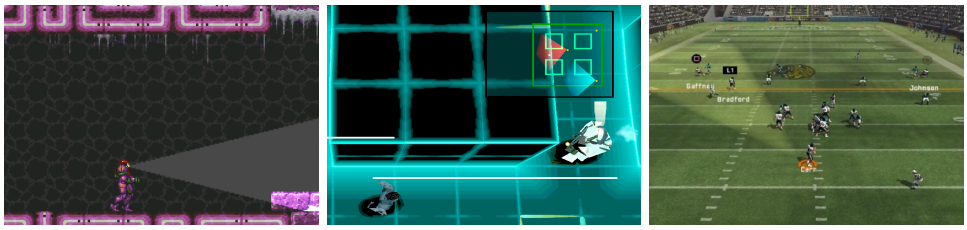


Figure 36: Three examples of vision cones in video games. From left to right these are from the 1994 Nintendo game *Super Metroid*, the 1998 Konami game *Metal Gear Solid* and the 2006 EA Sports game *Madden NFL 06*.

mented reality but it might also prove useful for purely virtual reality (VR) applications. While there are currently no plans to take any steps into that direction ourselves, it might apply even more, as there are of course no *natural* communication channels in VR. They could be simulated but this is probably difficult. Exploring channels that come easier to machines while being intuitive and unobtrusive for humans might be a common interest for both AR and VR researchers. A kind of purely virtual environment (although not necessarily VR) in which gaze direction displays have been used are video games. As Figure 36 exemplifies, vision cones are probably the most prevalent type. We can only speculate whether this is because they simply are the most intuitive and most informative type or because there is a lack of alternatives. It is somehow doubtful, though, that they are perfect for every application and they probably are not the least obtrusive display variety.

Better hardware and software

As pointed out in Chapter 1 and by many participants of our study, the uncomfortableness of the goggles and the lag and the reliability of the image recognition are major issues. While the advancement of AR hardware and image recognition algorithms is beyond the scope of C5 or any project derived thereof, there might be room for improvements of LAFORGE's performance.

Up to now, the priority of the visualization software development has rather been on the setup of the artificial communication channels than their computational efficiency. Profiling the software would reveal where time and energy can be fruitfully employed. What is probably only a small fraction of the overall computing time, yet a part of the program very easy to improve, is the XML parser. Xmltio builds a DOM tree from every packet received.

This is a waste of resources and a SAX or pull parser combined with the perfect minimal hash function already in place would probably have a fairly large effect on this fraction of the code.

Ambient intelligence

At the time being, the display techniques are quite prominent in the users' perception. This might change with habituation but one can also think of how the system itself could become more subtle. A philosophy called *ambient intelligence* treats software systems that have no clear-cut incarnation like a user interface or an agent. Instead, the intelligent system is "just there", constantly working in the background but only to become visible when such help is likely to be appreciated. Artificial communication channels may well turn out to be such a means by which an ambient system can subtly act upon the user. It might switch them on when the user is likely to seek conversation and off when he is not, it might change their parameters according to the situation and it might even use the same channels to augment its communication with the user. Objects that are referred to by other means like language might light up just as objects that are looked at by a person do. Furthermore, an advanced system might guess the user's intentions and point out some areas or objects of likely interest to him, thereby guiding his attention in an unobtrusive way.

In summary, the work at hand represents a solid basis for future interaction technology including mobile assistance systems, social and collaborative contexts, and communication research. It is difficult to predict how such techniques will change human interaction in the future but with decreasing weight of the gear the applicability of the presented methods will doubtlessly increase.

Bibliography

G. Aumann and K. Spitzmüller. *Computerorientierte Geometrie*, volume 89 of *Reihe Informatik*, chapter Projektionen, pages 203, 204, 215–217. B.I. Wissenschaftsverlag, 1993.

Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, and Blair MacIntyre. Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, 21(6):34–47, 2001. URL `http://www.cs.unc.edu/~azuma/cga2001.pdf`.

Ronald T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments (1054-7460)*, 6(4):355–385, 1997. URL `http://www.cs.unc.edu/~azuma/ARpresence.pdf`.

M. Billinghurst and H. Kato. Collaborative augmented reality, 2002. URL `citeseer.ist.psu.edu/billinghurst02collaborative.html`.

T.P. Caudell and D.W. Mizell. Augmented reality: an application of heads-up display technology to manual manufacturing processes. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, volume ii, pages 659–669, January 1992.

D. Drascic and P. Milgram. Perceptual issues in augmented reality. *SPIE*, 2653:123–134, 1996.

Christof Elbrechter. *Neuroinformatics Group Wiki – Vision Main Page*. Neuroinformatics Group Bielefeld University. URL `https://niwww.techfak.uni-bielefeld.de/wiki/index.php/VISION:Main_Page`.

Marc Hanheide. *A Cognitive Ego-Vision System for Interactive Assistance*. PhD thesis, Bielefeld University, 2006.

G. Heidemann, I. Bax, H. Bekel, C. Bauckhage, S. Wachsmuth, G. Fink, A. Pinz, H. Ritter, and G. Sagerer. Multimodal interaction in an augmented reality scenario. In *Proc. Int'l Conf. Multimodal Interfaces*, pages 53–60,

New York, NY, USA, 2004. ACM Press. URL http://www.bax.at/pdf/HeidemannEtAl-2004-ICMI.pdf.

Thomas Hermann. *Sonification for Exploratory Data Analysis*. PhD thesis, Bielefeld University, Bielefeld, Germany, February 2002. URL http://www.techfak.uni-bielefeld.de/ags/ni/publications/media/Hermann2002-SFE.pdf.

Henrik Hobein. Entwicklung eines online-kalibrierenden systems zur 3d-positionsbestimmung. Diplomarbeit, Bielefeld University, Neuroinformatics Group, 2007.

H. Kato and M. Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, 99:85–94, 1999.

K. Kiyokawa, Y. Kurata, and H. Ohno. An optical see-through display for mutual occlusion of real and virtual environments. In *Proc. IEEE and ACM International Symposium on Augmented Reality (ISAR 2000)*, pages 60–67, October 2000.

R.J. Kosinski. A literature review on reaction time. *On-line document*, 2002. URL http://biae.clemson.edu/bpc/bp/Lab/110/reaction.htm.

G. Kramer, B. Walker, T. Bonebright, P. Cook, J. Flowers, N. Miner, J. Neuhoff, et al. Sonification report: Status of the field and research agenda. *Report prepared for the National Science Foundation by members of the International Community for Auditory Display. Santa Fe, NM: International Community for Auditory Display (ICAD)*, 1999. URL http://www.icad.org/websiteV2.0/References/nsf.html.

Mark L. Knapp and Judith A. Hall. *Nonverbal Communication in Human Interaction*. Wadsworth/Thomson Learning, 5 edition, 2001. ISBN 9780155063723.

B.S. Majewski, N.C. Wormald, G. Havas, and Z.J. Czech. A family of perfect hashing methods. *The Computer Journal*, 39(6):547–554, 1996. URL http://www.math.uwaterloo.ca/~nwormald/papers/hashcompj.ps. Source code: http://www.ibiblio.org/pub/Linux/devel/lang/c/mph-1.2.tar.gz.

Paul Milgram and Fumio Kishino. A taxonomy of mixed reality visual displays. *IEICE Transactions on Information Systems*, E77-D(12):1321–1329, December 1994.

M.J. Pickering and S. Garrod. Toward a mechanistic psychology of dialogue. *Behavioral and Brain Sciences*, 27(02):169–190, 2004.

G. Rickheit, G. Jäger, H. Ritter, G. Sagerer, I. Wachsmuth, H. Strohner, S. Wachsmuth, L. Sichelschmidt, S. Kopp, H. Rieser, H.M. Müller, A. Mehler, M. Hielscher-Fastabend, H.-J. Eikmeyer, R. Weingarten, W. Kindt, J. Steil, and T. Hermann. Proposal for the establishment and funding of collaborative research centre 673 "alignment in communication" for 2006/2–2010/1, December 2005.

OpenGL 2.1 Reference Pages – gluLookAt. Silicon Graphics, Inc., 2006. URL `http://www.opengl.org/sdk/docs/man/xhtml/gluLookAt.xml`.

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. *Mixed and Augmented Reality, 2002. ISMAR 2002. Proceedings. International Symposium on*, pages 267–268, 2002. URL `http://eyetap.org/papers/docs/eye2eye.pdf`.

K.P. Truong and D.A. Leeuwen. Automatic detection of laughter. In *Ninth European Conference on Speech Communication and Technology*. ISCA, 2005.

R. Vertegaal. The gaze groupware system: mediating joint attention in multiparty communication and collaboration. *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, pages 294–301, 1999.

J. M. Wolfe and T. S. Horowitz. What attributes guide the deployment of visual attention and how do they do it? *Nature Reviews Neuroscience*, 5 (6):496–501, 2004.

Sebastian Wrede, Jannik Fritsch, Christian Bauckhage, and Gerhard Sagerer. An xml based framework for cognitive vision architectures. *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, 1, 2004.

M. Wright and A. Freed. Open sound control: A new protocol for communicating with sound synthesizers. *Proceedings of the 1997 International Computer Music Conference*, pages 101–104, 1997.

Z. Zeng, M. Pantic, G.I. Roisman, and T.S. Huang. A survey of affect recognition methods: audio, visual and spontaneous expressions. *Proceedings of the ninth international conference on Multimodal interfaces*, pages 126–133, 2007.

Appendices

A Software Installation

This appendix tells you everything you need to know to build the software presented in this work. If you have access to the wiki of the Neuroinformatics group of Bielefeld University, you can find the same information at `https://niwww.techfak.uni-bielefeld.de/wiki/index.php/C5:FOV_display_(usage)` in a form better tailored to the setup of that group.

If you have further questions or if you read this without having access to either the Bielefeld University intranet or the accompanying DVD, please do not hesitate to contact me by writing an email to `cmertes@techfak.uni-bielefeld.de`.

A.1 XCF

For PICARD you need `xcf4j`. The 1.1 branch was used which can be checked out from `https://xcf.svn.sourceforge.net/svnroot/xcf/xcf4j/branches/rel_1_1_maint`. You also need an C++ XCF implementation recent enough to use `spread` in order to build LAFORGE. For further documentation concerning XCF please refer to `http://sourceforge.net/projects/xcf/`. After building `xcf4j` with `ant`, you will find a file named `xcf4j-<date>.jar` in the `lib/` directory. This will be referred to as `xcf4j.jar` in the following so it makes sense to create a symlink.

On every computer you want to use XCF software on, you have to create a `.xcfrc` file containing a line `XCF.Initial.Host=<hostname>`, replacing `<hostname>` by the name of the machine the *dispatcher* will run on (see the usage hints below).

A.2 ICL

Please refer to the installation instructions at `https://niwww.techfak.uni-bielefeld.de/wiki/index.php/VISION:ICL` or use the version on the DVD as it is.

A.3 PICARD

Check out `svn+ssh://priamos.techfak.uni-bielefeld.de/vol/align/c5/share/repos/picard/trunk` and make sure your `CLASSPATH` contains `xcf4j.jar` and `NetUtil.jar` which can be obtained from `http://sourceforge.net/projects/netutil`. Then change to the directory you downloaded PICARD to and execute

```
javac de/unibi/techfak/romulus/picard/*.java
```

A.4 LAFORGE

Make sure your ICL installation is correct and all due paths are set accordingly, especially `ICL_ROOT` and `ICL_INSTALL_ROOT`. Check out `svn+ssh://priamos.techfak.uni-bielefeld.de/vol/ni/src/svnroot/vision/ICLProjects/trunk`, change to the directory and then execute the following:

```
cd ICLART
make depend all installlink
cd ../ICLLaforge
```

You can now edit the file `makeVar` and change `DEBUG_ON:=FALSE` to `DEBUG_ON:=TRUE` to enable assertions and more warnings. You can then also choose one of the "debugging flags" lines below the `DEBUG_ON` option. The one with `-DLAFORGE_NOVIDEOGRAB` allows you to debug the program without cameras attached. Finally you will want to execute

```
make depend all
```

or

```
make depend all 2>&1 | ./filterwarnings.py
```

if you enabled the warnings and do not want to see the whole lot of them XCF produces.

You finally have to set the environment variable `ART_PATTERN_PATH` to `$ICLPROJECTS/ICLART/Data/usedPatterns` where `$ICLPROJECTS` is a placeholder for the directory you checked the above repository out to.

A.5 MILES

Check out `svn+ssh://priamos.techfak.uni-bielefeld.de/vol/align/c5/share/repos/mt9/branches/altsmooth`. Change to the directory and edit the `Makefile` if you want to switch off XCF or OSC support. You can do so by removing the `1` after the `USEXCF = ` or `USEOSC = ` definition respectively. You might also have to adjust path names in the `Makefile` and the file `miles` which is a wrapper script for `mt9`. If you use MT9-A inertial sensors, you need a calibration file which usually has a `.xmu` file name extension. Create a symbolic link to this file named `calib.xmu`.[29] To build just execute `make`.

A.6 SuperCollider

Get SuperCollider from `http://www.audiosynth.com/` and install it according to the instructions found there. The `sclang` mode for `emacs` or the *sced* plugin[30] for `gedit` also come in handy. You only need to run `sclang` once, though. You need to store the synths on disc in order to allow `scsynth` to play them later when receiving the OSC command to do so. The continuous sonification uses the synth `\fovSon`, the discrete one the synths `\objIn` and `\objOut`. To produce the synths described in Section 2.2 you need to execute in sclang the following two lines of code:

```
SynthDef(\objIn, {
        Out.ar(0, SinOsc.ar(880) *
        EnvGen.kr(Env.perc, doneAction: 2, timeScale: 0.6))
}).store;
SynthDef(\objOut, {
        Out.ar(0, (SinOsc.ar(440, 0, 1) + PinkNoise.ar(0.3)) *
```

[29] Actually you can use any `.xmu` file in the MILES home directory as long as it is the only one.

[30] `http://artfwo.googlepages.com/sced`

```
        EnvGen.kr(Env.perc, doneAction: 2, timeScale: 0.3))
}).store;
```

After that, scsynth should run out of the box. jack might still pose problems, though. For the hardware described in Section 3.1 we needed the following $HOME/.jackdrc:

```
/usr/bin/jackd -R -dalsa -dhw:0 -r44100 -p128 -n4
```

or the sound would stutter heavily despite the real-time kernel. Finally you need to specify the host and port the scsynth for system 1 and 2 run on by setting the LAFORGE_SC_HOST1 and LAFORGE_SC_HOST1 environment variables in the usual <host>[:<port>] style. If no port is given, 57110 is used.

A.7 Wii Software

First you need the tool xcfWiiAdapter written by Marc Hanheide. You can check it out from https://code.ai.techfak.uni-bielefeld.de/scm/ai/trunk/xcfWiiAdapter. To build, execute

```
rm -rf CMakeFiles/ CMakeCache.txt
ccmake .
make
```

in the xcfWiiAdapter directory. You then need the *Wiibrations* tool, whose repository is located at svn+ssh://priamos/vol/align/c5/share/repos/wiibrations/trunk. Change to its directory and execute

```
javac de/unibi/techfak/romulus/wiibrator/*.java
```

B Usage Information

B.1 Using the GUI

The easy way to start the components of the system that you need is by using the Python script `start.py`. You will have to go through it and change paths, environment variables and host names where needed and you might want to change the directory the text output is written to. But once you did that, you can simply execute this script and start and stop the programs you need with the click of a button.

Below all other buttons is what is called the *trial counter*. By pressing the *next* button, you can increment a variable whose value is added to every file name of the output files, along with the time and date of the button press in a human-readable format and in milliseconds since 1970. This only applies for new files, so you have to stop and restart data-producing programs in order for the increment to have an effect. Figure 37 shows a screenshot of a vertically shortened `start.py` instance.

B.2 Manual Execution

Comfortable as it is to use the GUI, it might be necessary or educational to know how the various tools are executed manually.

For XCF to work, you first of all need to execute the so called *dispatcher*. For `xcf4j` this is done with

```
java -jar xcf4j.jar
```

while `xcf4j.jar` still refers to the file created by building `xcf4j`. You might want to define an alias to this command using an absolute path name to

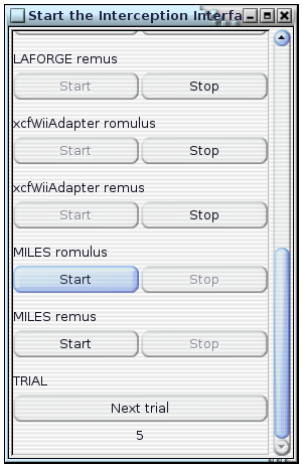


Figure 37: Screenshot of the graphical startup script `start.py`. *romulus* and *remus* are the names of the two laptop computers our system runs on.[31] You will want to adapt this to your setup.

this file. You have to start the dispatcher on only one machine – the one you set in your `.xcfrc` (see above).

On any one computer, go to the PICARD directory and execute

```
java de.unibi.techfak.romulus.picard.Main --gui --highlight
```

You can add the `java` option `-ea` to enable assertions in PICARD. The `--gui` option is not required but you will not be able to change any settings during run-time if you run PICARD without its GUI. To get a list of options that control its start-up settings, run PICARD with `--help`. See Figure 18 for a screenshot of the PICARD GUI.

The last tool you have to start only once is the Wii GUI for task 2 of our experiment (Section 4.1). If you need this, go to the *Wiibrations* directory and execute

```
java de.unibi.techfak.romulus.wiibrations.Main > $WIIDATAFILE
```

`$WIIDATAFILE` is the file you want your Wii data in.

The following commands you have to execute once per HMD. We use two laptops but in principle with the right configuration it should work with only one computer. Getting both augmented video streams out would probably

[31] As you are probably a curious person if you read this far: These names are not *primarily* part of the naming scheme from Box 1 but rather the scheme prescribing moon names for laptops in our group. So for our twin computers this pair came in handy (`http://apod.nasa.gov/apod/ap050818.html`).

be the greatest challenge and would require a more sophisticated X Window configuration.

In order to use sound, you need to start `scsynth`. You should do this in accordance with the environment variables you set but given you did not change the port and use two computers, you would execute

```
sudo scsynth -u 57110
```

on both of them. If you have trouble here, let me recommend an ugly workaround that works like a charm for me: Run `gedit` (as superuser, if needed) with the *sced* plugin, choose "SuperCollider Mode" from the "Tools" menu and from the then appearing "SuperCollider" menu choose "Start Server".

In the following, <1|2> should be replaced with either 1 or 2, depending on which HMD this instance shall be responsible for (so normally which computer it is running on). This is called the *system ID*. In the LAFORGE directory, execute

```
dc1394_reset_bus &&
./application/laforge -xcf -text -lt -sytemID <1|2>\
   > $LAFORGEDATAFILE<1|2>
```

It is important not to `cd` into `application/` but into its parent directory. LAFORGE can be used with the `--help` option, too. The `dc1394_reset_bus` is not strictly necessary but the *libdc*[32] more or less forces this habit upon you of executing the command every time you access the FireWire cameras.

When using the Wiis, change to the `xcfWiiAdapter` directory and call

```
./xcfWiiAdapter -a false -p WiiData<1|2> -s WiiFeedback<1|2>
```

Do not do this at the same time for both systems, as the tool will ask you to press buttons 1 and 2 on the Wii Remote simultaneously. This is needed to associate the Wii Remote with the instance of `xcfWiiAdapter` so you must wait until the synchronization is complete before you proceed with the second system. The `xcfWiiAdapter` understands `--help`.

Finally, if you want to use the inertial sensors, `cd` to the MILES directory and call

```
./miles --text --raw --time -d, --out $MILESDATAFILE<1|2>
```

MILES, too, will tell you more about its usage when called with `--help`.

[32] `http://sourceforge.net/projects/libdc1394/`

C Doxygen Documentation

This appendix includes documentation that was created automatically from the sources of PICARD and LAFORGE using the tool Doxygen[33]. This may serve as an excuse for the comparatively poor typesetting and the inconsistencies between this appendix and the rest of this work. I hope you will find the following sections insight- as well as helpful nonetheless.[34]

C.1 PICARD Class Documentation

CalculateHelper Class Reference

Detailed Description

This class provides several mathematical functions that can be used by other classes.

Author:

cmertes

Static Public Member Functions

– static double[] xyIntersection (double[] pos, double[] normal)

Calculates the intersection of a given ray with the plane E_{12} *spanned by the* x *and* y *axes.*

[33] `http://www.doxygen.org/`

[34] If you prefer browsing the source documentation in HTML, you might want to have a look at the file `doc/html/classes.html` in the root directory of PICARD or LAFORGE respectively.

- static double[] xzIntersection (double[] pos, double[] normal)

  Calculates the intersection of a given ray with the plane $\mathbb{E}_{13}$ *spanned by the* x *and* z *axes.*

- static double[] xyProjection (double[] vec)

  Calculates the parallel projection of a given n-dimensional vector onto the plane $\mathbb{E}_{12}$ *spanned by the* x *and* y *axes.*

- static double vecDist (double[] pos1, double[] pos2)

  Calculates the distance between two points in n*-dimensional Euclidean space.*

- static double dotProduct (double[] vec1, double[] vec2)

  Calculates the scalar product of the two input vectors.

- static double[] crossProduct (double[] v1, double[] v2)

  Calculates the cross product of two three dimensional vectors in respect to a right-handed coordinate system.

- static double abs (double[] vec)

  Calculates the magnitude of a vector, i.e. its length in Euclidean space.

- static void normalize (double[] vec)

  Normalises the vector it gets passed.

- static double[] vecDiff (double[] vec1, double[] vec2)

  Subtracts one vector from another, returning vec1 − vec2*.*

- static double cosOfAngle (double[] vec1, double[] vec2)

  Calculates the cosine of the angle between two vectors.

- static double angleDirection (double[] vec1, double[] vec2)

  Uses the z *component of the cross product of two vectors to determine whether the angle between the two vectors is clockwise or counterclockwise.*

- static double sig (double d)

  A sigmoid function.

- static double mapAroundZero (double d, double bound)

  Maps any value d *to [*-bound, bound*] in a sigmoid manner using the hyperbolic tangent.*

– static double mapTo (double d, double lowerBound, double upperBound)

  Uses the sig function to map any value to [`lowerBound`*,* `upperBound`*].*

– static double mapAroundZero (double d, double bound, double factor, double offset)

  Maps any value `d` *to [-*`bound`*,* `bound`*] in a sigmoid manner using the hyperbolic tangent.*

– static double mapTo (double d, double lowerBound, double upperBound, double factor, double offset)

  Uses the sig function to map any value to [`lowerBound`*,* `upperBound`*].*

– static int nearestExtreme (int value, int minimum, int maximum)

  This function returns either `minimum` *or* `maximum`*, depending on which one is closer to* `value`*.*

Member Function Documentation

static double [] CalculateHelper.xyIntersection (double[] *pos*, double[] *normal*) `[static]`

Calculates the intersection of a given ray with the plane E_{12} spanned by the *x* and *y* axes.

Parameters:

pos The position vector of the ray's starting point.

normal The normalized direction vector of the ray.

Returns:

A two-dimensional array containing the *x* and *y* coordinates of the intersection. The *z* coordinate is always 0 and thus omitted. For a ray parallel to the plane `null` is returned.

static double [] CalculateHelper.xzIntersection (double[] *pos*, double[] *normal*) `[static]`

Calculates the intersection of a given ray with the plane E_{13} spanned by the *x* and *z* axes.

Parameters:

pos The position vector of the ray's starting point.

normal The normalized direction vector of the ray.

Returns:

A two-dimensional array containing the *x* and *z* coordinates of the intersection. The *y* coordinate is always 0 and thus omitted. For a ray parallel to the plane `null` is returned.

static double [] CalculateHelper.xyProjection (double[] *vec*) `[static]`

Calculates the parallel projection of a given n-dimensional vector onto the plane E_{12} spanned by the *x* and *y* axes.

Parameters:

vec The vector to project.

Returns:

The resulting two-dimensional vector.

static double CalculateHelper.vecDist (double[] *pos1*, double[] *pos2*) `[static]`

Calculates the distance between two points in *n*-dimensional Euclidean space.

n being the lesser of the two dimensions of the input vectors.

Parameters:

pos1 Position vector of the first point.

pos2 Position vector of the second point.

Returns:

The Euclidean distance between the points specified by the two input vectors.

static double CalculateHelper.dotProduct (double[] *vec1*, double[] *vec2*) `[static]`

Calculates the scalar product of the two input vectors.

The vectors can be of any dimension but for vectors with different dimensions excess values are ignored.

Parameters:

vec1 First input vector.

vec2 Second input vector.

Returns:

The dot product of the two input vectors.

static double [] CalculateHelper.crossProduct (double[] *v1*, double[] *v2*) `[static]`

Calculates the cross product of two three dimensional vectors in respect to a right-handed coordinate system.

Parameters:

v1 first input vector

v2 second input vector

Returns:

the cross product v1 $\times$ v2 of the two input vectors

static double CalculateHelper.abs (double[] *vec*) `[static]`

Calculates the magnitude of a vector, i.e. its length in Euclidean space.

Parameters:

vec The input vector.

Returns:

The magnitude ‖vec‖ of the input vector.

static void CalculateHelper.normalize (double[] *vec*) `[static]`

Normalises the vector it gets passed.

Parameters:

vec The vector to normalise

static double [] CalculateHelper.vecDiff (double[] *vec1*, double[] *vec2*) `[static]`

Subtracts one vector from another, returning vec1 − vec2.

Parameters:

vec1 The minuend.

vec2 The subtrahend.

Returns:

The difference between the two input vectors.

static double CalculateHelper.cosOfAngle (double[] *vec1*, double[] *vec2*) `[static]`

Calculates the cosine of the angle between two vectors.

So `Math.acos` of this method's result would be the angle between two vectors. The arccosine isn't precalculated because this method's return value can be used for further calculations itself.

Parameters:

vec1 The first input vector.

vec2 The second input vector.

Returns:

The cosine of the angle between the two input vectors, that is $\frac{\mathrm{vec1} \cdot \mathrm{vec2}}{\|\mathrm{vec1}\|\|\mathrm{vec2}\|}$.

static double CalculateHelper.angleDirection (double[] *vec1*, double[] *vec2*) `[static]`

Uses the *z* component of the cross product of two vectors to determine whether the angle between the two vectors is clockwise or counterclockwise.

Parameters:

vec1 An at least two-dimensional input vector.

vec2 An at least two-dimensional input vector.

Returns:

-1.0 or 1.0, determined by $\mathrm{sgn}\,(\mathrm{vec1}_1\mathrm{vec2}_2 - \mathrm{vec1}_2\mathrm{vec2}_1)$.

static double CalculateHelper.sig (double *d*) `[static]`

A sigmoid function.

The logistic function $\mathrm{sig}(\mathrm{d}) = \frac{1}{1+e^{-\mathrm{d}}}$ is used. This function is bounded below by 0 and bounded above by 1.

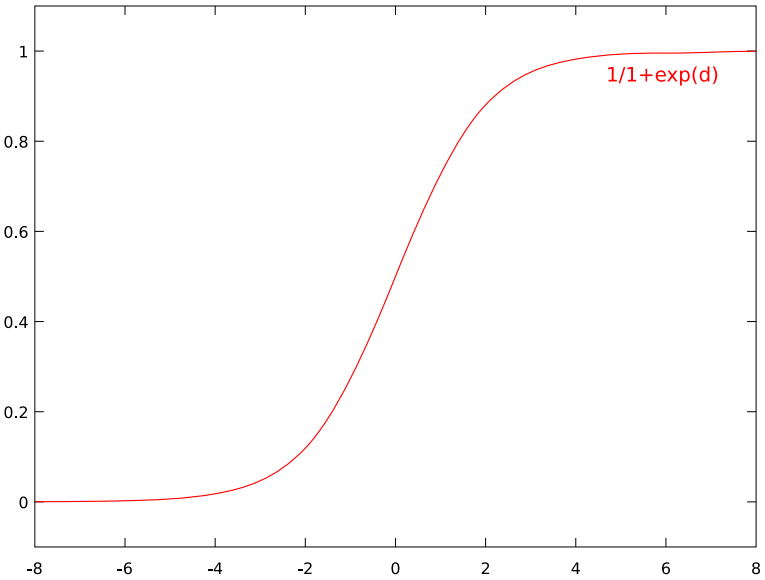


Figure 38: Sigmoid function plot

Parameters:

d The function parameter.

Returns:

The result of sig(d) as described above.

static double CalculateHelper.mapAroundZero (double *d*, double *bound*) `[static]`

Maps any value d to [-bound, bound] in a sigmoid manner using the hyperbolic tangent.

Parameters:

d The value to be mapped.

bound The bound to be mapped into.

Returns:

A value between -bound and bound, calculated by tanh(d)bound.

See also:

mapAroundZero(double, double, double, double)

static double CalculateHelper.mapTo (double *d*, double *lowerBound*, double *upperBound*) `[static]`

Uses the sig function to map any value to [lowerBound, upperBound].

Parameters:

d The value to be mapped.

lowerBound The lower bound of the mapping range.

upperBound The upper bound of the mapping range.

Returns:

$\text{sig}(\texttt{d})(\texttt{upperBound} - \texttt{lowerBound}) + \texttt{lowerBound}$

See also:

mapTo(double, double, double, double, double)

static double CalculateHelper.mapAroundZero (double *d*, double *bound*, double *factor*, double *offset*) `[static]`

Maps any value d to [-bound, bound] in a sigmoid manner using the hyperbolic tangent.

Before mapping `offset` is added to d and the result is multiplied by `factor`. This means `offset` influences the zero of the function and `factor` its slope.

Parameters:

d The value to be mapped.

bound The bound to be mapped into.

factor The factor by which d is multiplied.

offset The offset added to d.

Returns:

tanh((d + offset)factor)bound

See also:

mapAroundZero(double, double)

static double CalculateHelper.mapTo (double *d*, double *lowerBound*, double *upperBound*, double *factor*, double *offset*) [static]

Uses the sig function to map any value to [lowerBound, upperBound].

Before mapping offset is added to d and the result is multiplied by factor. This means offset influences the zero of the function and factor its slope.

Parameters:

d The value to be mapped.

lowerBound The lower bound of the mapping range.

upperBound The upper bound of the mapping range.

factor The factor by which d is multiplied.

offset The offset added to d.

Returns:

sig((d + offset)factor)(upperBound − lowerBound) + lowerBound

See also:

mapTo(double, double, double)

static int CalculateHelper.nearestExtreme (int *value*, int *minimum*, int *maximum*) [static]

This function returns either minimum or maximum, depending on which one is closer to value.

If both extremes are equally close to `value`, `maximum` is returned. If `value` is not within [`minimum`, `maximum`], the result is undefined.

Parameters:

value The value which must be between `minimum` and `maximum` and which determines the extreme to be chosen.

minimum The smaller extreme.

maximum The greater extreme.

Returns:

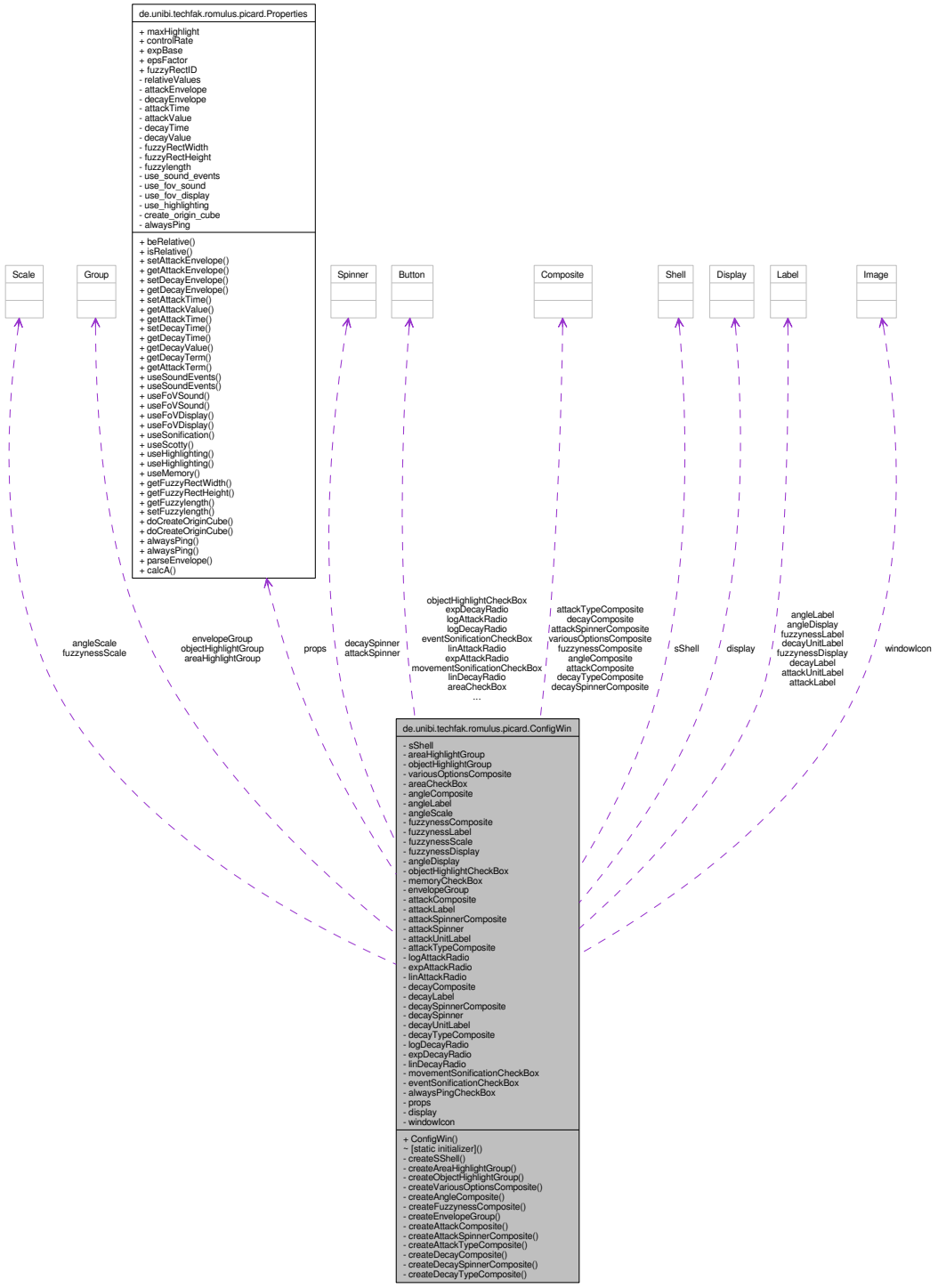
`minimum` or `maximum`, whichever is closer to `value`.

The documentation for this class was generated from the following file:

– CalculateHelper.java

ConfigWin Class Reference

Collaboration diagram for ConfigWin:



Detailed Description

This is the main window controlling all important parameters for both LAFORGE systems that can be connected to the controlling PICARD software.

Author:

cmertes

Public Member Functions

- ConfigWin (Properties properties)
  The class constructor.

Static Package Functions

- **[static initializer]**

Private Member Functions

- void createSShell ()
  This method initializes sShell.
- void createAreaHighlightGroup ()
  This method initializes areaHighlightGroup.
- void createObjectHighlightGroup ()
  This method initializes objectHighlightGroup.
- void createVariousOptionsComposite ()
  This method initializes variousOptionsComposite.
- void createAngleComposite ()
  This method initializes angleComposite.
- void createFuzzynessComposite ()
  This method initializes fuzzynessComposite.

- void createEnvelopeGroup ()
  This method initializes envelopeGroup.

- void createAttackComposite ()
  This method initializes attackComposite.

- void createAttackSpinnerComposite ()
  This method initializes attackSpinnerComposite.

- void createAttackTypeComposite ()
  This method initializes attackTypeComposite.

- void createDecayComposite ()
  This method initializes decayComposite.

- void createDecaySpinnerComposite ()
  This method initializes decaySpinnerComposite.

- void createDecayTypeComposite ()
  This method initializes decayTypeComposite.

Private Attributes

- Shell **sShell** = null
- Group **areaHighlightGroup** = null
- Group **objectHighlightGroup** = null
- Composite **variousOptionsComposite** = null
- Button **areaCheckBox** = null
- Composite **angleComposite** = null
- Label **angleLabel** = null
- Scale **angleScale** = null
- Composite **fuzzynessComposite** = null
- Label **fuzzynessLabel** = null
- Scale **fuzzynessScale** = null
- Label **fuzzynessDisplay** = null
- Label **angleDisplay** = null
- Button **objectHighlightCheckBox** = null
- Button **memoryCheckBox** = null

- Group **envelopeGroup** = null
- Composite **attackComposite** = null
- Label **attackLabel** = null
- Composite **attackSpinnerComposite** = null
- Spinner **attackSpinner** = null
- Label **attackUnitLabel** = null
- Composite **attackTypeComposite** = null
- Button **logAttackRadio** = null
- Button **expAttackRadio** = null
- Button **linAttackRadio** = null
- Composite **decayComposite** = null
- Label **decayLabel** = null
- Composite **decaySpinnerComposite** = null
- Spinner **decaySpinner** = null
- Label **decayUnitLabel** = null
- Composite **decayTypeComposite** = null
- Button **logDecayRadio** = null
- Button **expDecayRadio** = null
- Button **linDecayRadio** = null
- Button **movementSonificationCheckBox** = null
- Button **eventSonificationCheckBox** = null
- Button **alwaysPingCheckBox** = null
- Properties **props**

Static Private Attributes

- static Display **display**
- static Image **windowIcon**

Constructor & Destructor Documentation

ConfigWin.ConfigWin (Properties *properties*)

The class constructor.

It needs to be passed the global properties. This must be called last because this method never returns. When the window is closed, the program exits.

Parameters:

properties All configuration changes are stored here and the presets are read from this object.

Member Function Documentation

void ConfigWin.createAreaHighlightGroup () `[private]`

This method initializes areaHighlightGroup.

void ConfigWin.createObjectHighlightGroup () `[private]`

This method initializes objectHighlightGroup.

void ConfigWin.createVariousOptionsComposite () `[private]`

This method initializes variousOptionsComposite.

void ConfigWin.createAngleComposite () `[private]`

This method initializes angleComposite.

void ConfigWin.createFuzzynessComposite () `[private]`

This method initializes fuzzynessComposite.

void ConfigWin.createEnvelopeGroup () `[private]`

This method initializes envelopeGroup.

void ConfigWin.createAttackComposite () `[private]`

This method initializes attackComposite.

void ConfigWin.createAttackSpinnerComposite () `[private]`

This method initializes attackSpinnerComposite.

void ConfigWin.createAttackTypeComposite () `[private]`

This method initializes attackTypeComposite.

void ConfigWin.createDecayComposite () `[private]`

This method initializes decayComposite.

void ConfigWin.createDecaySpinnerComposite () `[private]`

This method initializes decaySpinnerComposite.

void ConfigWin.createDecayTypeComposite () `[private]`

This method initializes decayTypeComposite.

The documentation for this class was generated from the following file:

- ConfigWin.java

Idler Class Reference

Detailed Description

What do idlers do? Idle!

Author:

cmertes

Public Member Functions

- void idle ()

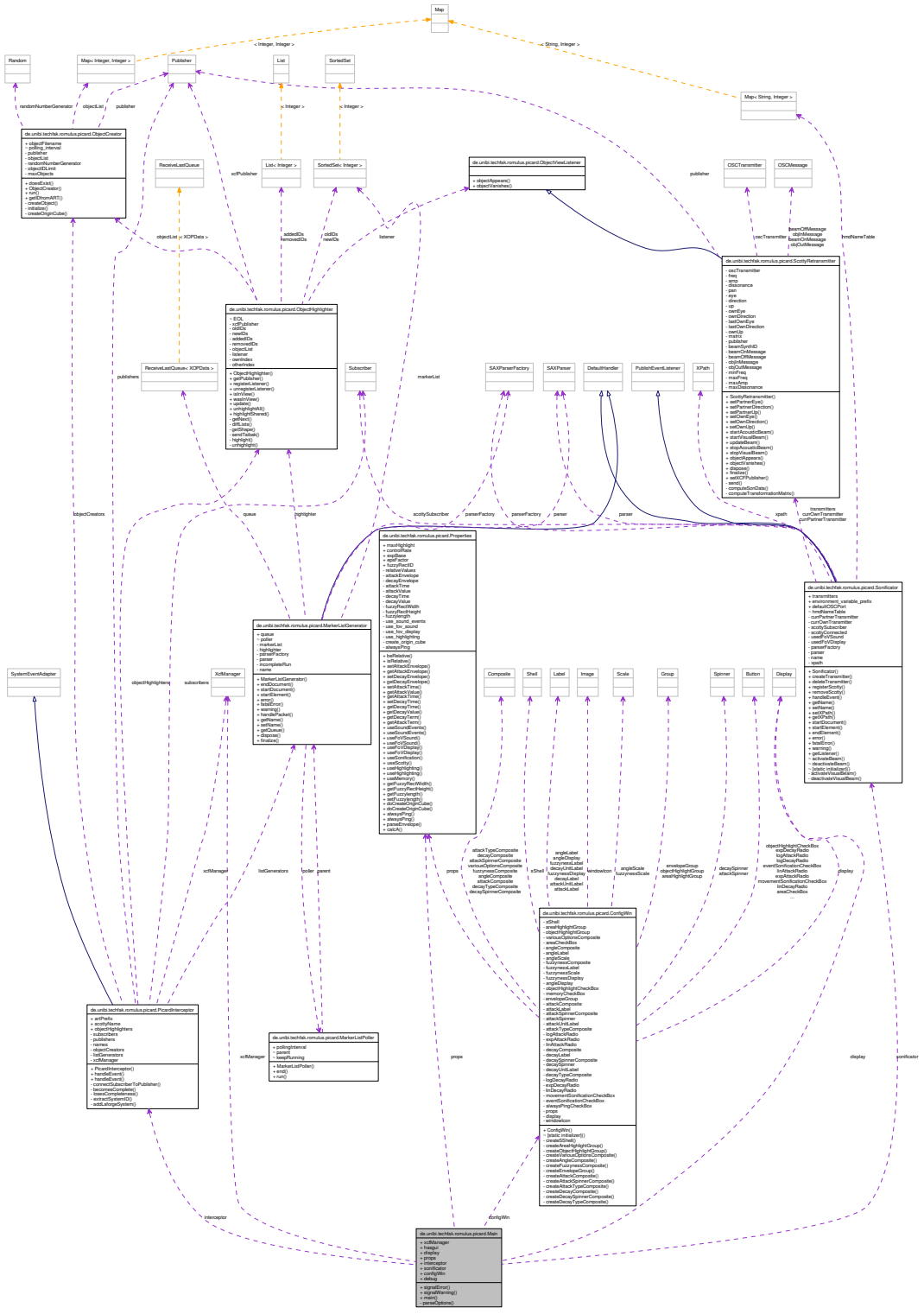
  Calls `wait()` *till the end of time.*

The documentation for this class was generated from the following file:

- Idler.java

Main Class Reference

Collaboration diagram for Main:



Detailed Description

This is the main class of PICARD, containing the main() method.

It parses the command line options, creates the main window if told so, creates an XcfManager and a Sonificator and instantiates PicardInterceptor, too, which in turn instantiates most other classes needed to run PICARD.

Author:

cmertes

Static Public Member Functions

- static synchronized void signalError (final String title, final String message)

  This method is used throughout the program to signal an error condition to the user.

- static synchronized void signalWarning (final String title, final String message)

  This method is used throughout the program to signal a non-critical abnormal condition to the user.

- static void main (String[] args)

  Program execution starts here.

Static Public Attributes

- static XcfManager **xcfManager**
- static boolean **hasgui**
- static Display **display**
- static Properties **props**
- static PicardInterceptor **interceptor**
- static Sonificator **sonificator**
- static ConfigWin **configWin**
- static final boolean **debug** = true

Static Private Member Functions

- static boolean parseOptions (String[] args)
  Writes a string to `autolastwords.log`.

Member Function Documentation

static synchronized void Main.signalError (final String *title*, final String *message*) [static]

This method is used throughout the program to signal an error condition to the user.

Depending on whether PICARD is running in GUI mode or not a message window is opened or the message is sent to `stderr`. The function does not terminate the program execution. If this is the desired behaviour after an unrecoverable error, the calling function must do so itself.

Parameters:

title A very brief phrase summarizing the error.

message A preferably verbose description of the error.

See also:

signalWarning(String, String)

static synchronized void Main.signalWarning (final String *title*, final String *message*) [static]

This method is used throughout the program to signal a non-critical abnormal condition to the user.

Currently the message is just sent to `stdout`. The program should never terminate after a warning.

Parameters:

title A very brief phrase summarizing the warning.

message A preferably verbose description of the problem.

See also:

signalError(String, String)

static boolean Main.parseOptions (String[] *args*) `[static, private]`

Writes a string to `autolastwords.log`.

Parameters:

message The string to write to the log file Parses the command line options and sets Properties accordingly.

args The `String` array passed to main.

Returns:

Whether the options could be parsed correctly.

static void Main.main (String[] *args*) `[static]`

Program execution starts here.

Parameters:

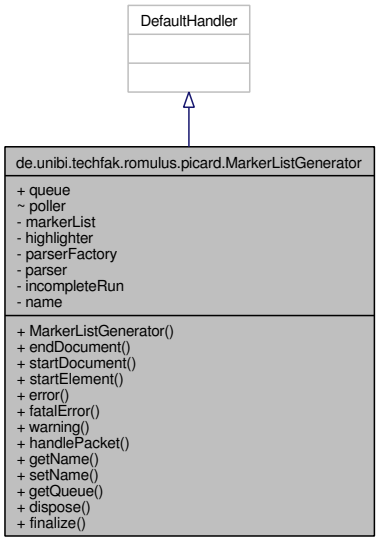
args The command line options passed to PICARD. Use `-help` for a summary.

The documentation for this class was generated from the following file:

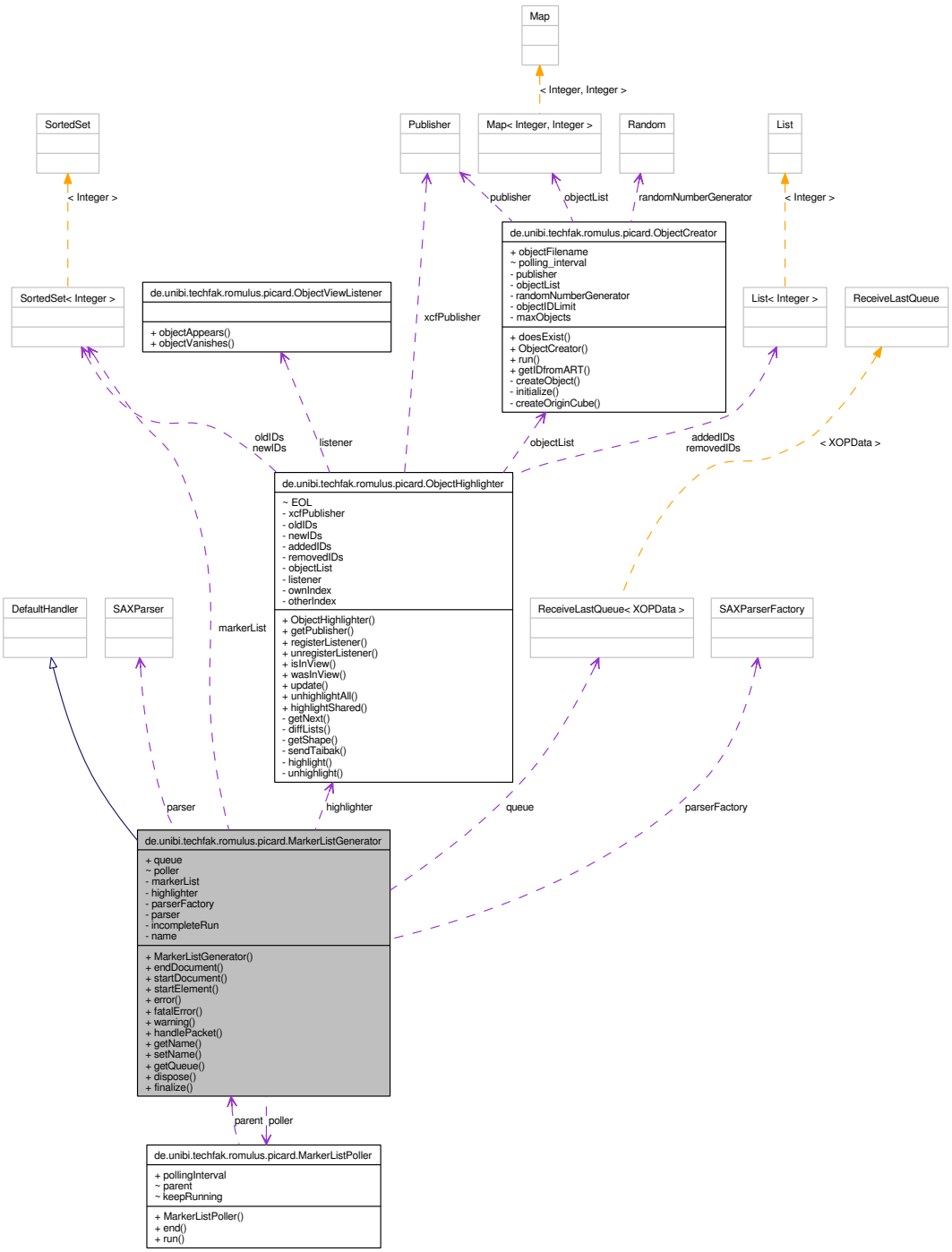
– Main.java

MarkerListGenerator Class Reference

Inheritance diagram for MarkerListGenerator:



Collaboration diagram for MarkerListGenerator:



Detailed Description

This class gets XML data packages containing the list of objects seen by a LAFORGE system and parses them using a SAX parser.

It then passes this list to its associated ObjectHighlighter instance. There has to be one `MarkerListGenerator` for every LAFORGE system.

Author:

cmertes

Public Member Functions

- MarkerListGenerator (ObjectHighlighter highlighter, ReceiveLastQueue< XOPData > queue)

  The class constructor.

- void **endDocument** ()
- void **startDocument** ()
- void **startElement** (String namespaceURI, String localName, String qName, Attributes atts)
- void **error** (SAXParseException exception)
- void **fatalError** (SAXParseException exception)
- void **warning** (SAXParseException exception)
- void handlePacket (XOPData data)

  This method is called for every new XML packet received.

- String **getName** ()
- void **setName** (String s)
- ReceiveLastQueue< XOPData > **getQueue** ()
- void **dispose** ()
- void **finalize** ()

Public Attributes

- ReceiveLastQueue< XOPData > queue

  Always contains the last XCF packet.

Package Attributes

- MarkerListPoller **poller**

Private Attributes

- SortedSet< Integer > **markerList** = null
- ObjectHighlighter **highlighter**
- SAXParserFactory **parserFactory**
- SAXParser **parser**
- boolean incompleteRun = false

  Debug variable to check for incomplete or concurrent calls to the parsing method.

- String **name**

Constructor & Destructor Documentation

MarkerListGenerator.MarkerListGenerator (ObjectHighlighter *highlighter*, ReceiveLastQueue< XOPData > *queue*)

The class constructor.

Parameters:

highlighter The ObjectHighlighter that the object lists get passed to.

queue The queue the new XCF packets can be found in.

Member Function Documentation

void MarkerListGenerator.handlePacket (XOPData *data*)

This method is called for every new XML packet received.

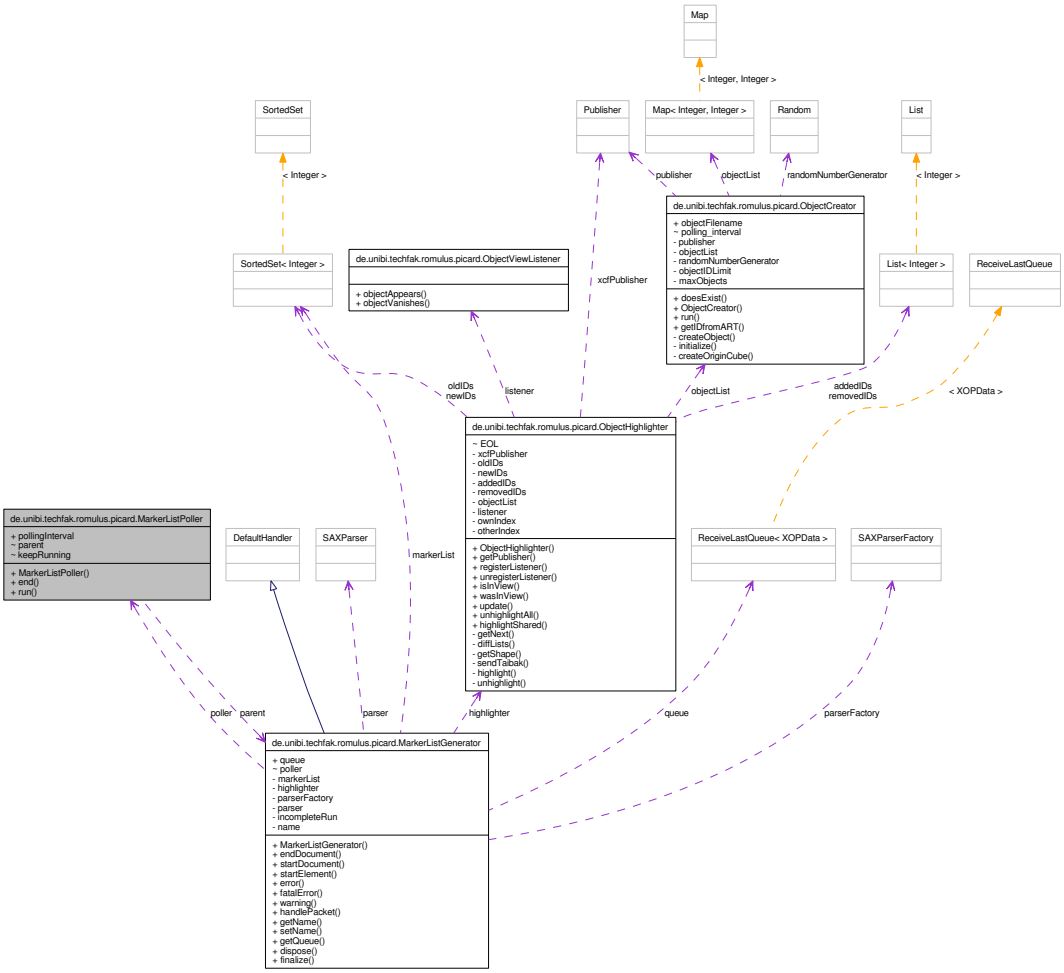
It is then parsed using this class itself.

The documentation for this class was generated from the following file:

– MarkerListGenerator.java

MarkerListPoller Class Reference

Collaboration diagram for MarkerListPoller:



Detailed Description

This class is a thread that checks the MarkerListGenerator.queue variable of its parent every 5 milliseconds and if a new XCF packet is available it triggers its parent to parse it via MarkerListGenerator.handlePacket(XOPData).

This is done to prevent too many data to come in. A solution without polling would be great though.

Author:

cmertes

Public Member Functions

- **MarkerListPoller** (MarkerListGenerator parent)
- void **end** ()
- void **run** ()

Static Public Attributes

- static final int **pollingInterval** = 5

Package Attributes

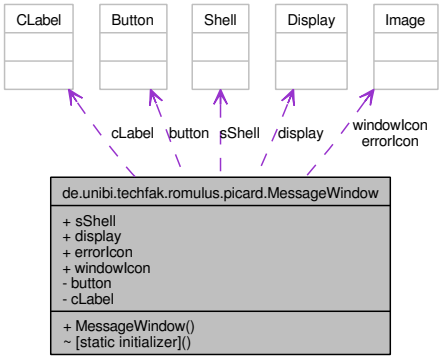
- MarkerListGenerator **parent**
- boolean **keepRunning** = true

The documentation for this class was generated from the following file:

- MarkerListPoller.java

MessageWindow Class Reference

Collaboration diagram for MessageWindow:



Detailed Description

A little helper class to display error messages to the user when the GUI is used instead of the console.

Author:

cmertes

Public Member Functions

- MessageWindow (String name, String message)

  The class constructor takes two strings.

Public Attributes

- Shell **sShell** = null

Static Public Attributes

- static Display **display**

- static Image **errorIcon**
- static Image **windowIcon**

Static Package Functions

- **[static initializer]**

Private Attributes

- Button **button** = null
- CLabel **cLabel** = null

Constructor & Destructor Documentation

MessageWindow.MessageWindow (String *name*, String *message*)

The class constructor takes two strings.

One to use as a window title and the other as the textual window content.

Parameters:

name The window title to use

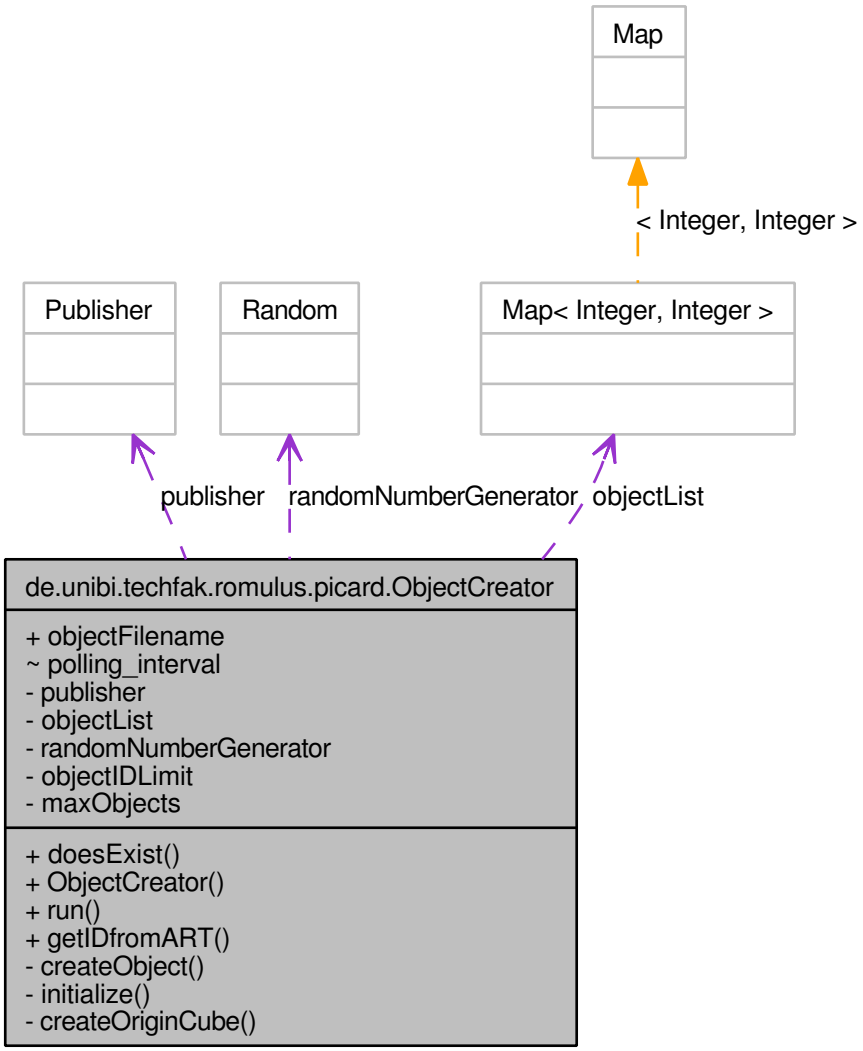
message The textual content of the message window

The documentation for this class was generated from the following file:

- MessageWindow.java

ObjectCreator Class Reference

Collaboration diagram for ObjectCreator:



Detailed Description

Author:

cmertes

Public Member Functions

- synchronized boolean **doesExist** (int objectID)
- ObjectCreator (Publisher newPublisher)

  The class constructor.

- void run ()

  When called polls regularly to see whether the newly started LAFORGE system is ready to receive our taibaks.

- synchronized Integer getIDfromART (Integer artID)

  Returns the object ID that belongs to a given ART ID.

Static Public Attributes

- static final String objectFilename = "object_uris"

  The name of the configuration file that contains the names of the mesh description files on the system running LAFORGE.

Static Package Attributes

- static final long polling_interval = 100

  At the beginning we have to wait until the LAFORGE system gets ready to receive our initialization taibaks.

Private Member Functions

- void createObject (int artID, String objFileURI)

  Sends a taibak to create a new mesh object.

- boolean initialize ()

  Initializes a single LAFORGE system2.

- void createOriginCube ()

  Mainly a debug function that creates a unit cube around the origin.

Private Attributes

- Publisher **publisher**
- Map< Integer, Integer > **objectList**

Static Private Attributes

- static Random **randomNumberGenerator** = new Random()
- static final int **objectIDLimit** = 10000
- static final int maxObjects = 6

  Generate no more objects than this.

Constructor & Destructor Documentation

ObjectCreator.ObjectCreator (Publisher *newPublisher*)

The class constructor.

Needs an XCF publisher to send the taibaks to that create the ART objects.

Parameters:

newPublisher The publisher to use

Member Function Documentation

void ObjectCreator.createObject (int *artID*, String *objFileURI*)
`[private]`

Sends a taibak to create a new mesh object.

Parameters:

artID The ART marker to create the object on

objFileURI The path to the file containing the object desciption of the mesh that shall be created

boolean ObjectCreator.initialize () `[private]`

Initializes a single LAFORGE system2.

Returns:

Currently always `true`. Errors in this function cause the program to abort.

void ObjectCreator.run ()

When called polls regularly to see whether the newly started LAFORGE system is ready to receive our taibaks.

Then it is initialized and this method begins to idle, the object waiting to answer questions about ART ID → object ID mappings. This method gets called by Thread.start().

synchronized Integer ObjectCreator.getIDfromART (Integer *artID*)

Returns the object ID that belongs to a given ART ID.

Parameters:

artID The ART ID to check for

Returns:

An `Integer` containing the object ID that can be used in taibaks to manipulate the object

Member Data Documentation

final String ObjectCreator.objectFilename = "object_uris" `[static]`

The name of the configuration file that contains the names of the mesh description files on the system running LAFORGE.

Each line is interpreted as one file name except when preceded by #or if there are more lines than the value ofmaxObjects.

final long ObjectCreator.polling_interval = 100 [static, package]

At the beginning we have to wait until the LAFORGE system gets ready to receive our initialization taibaks.

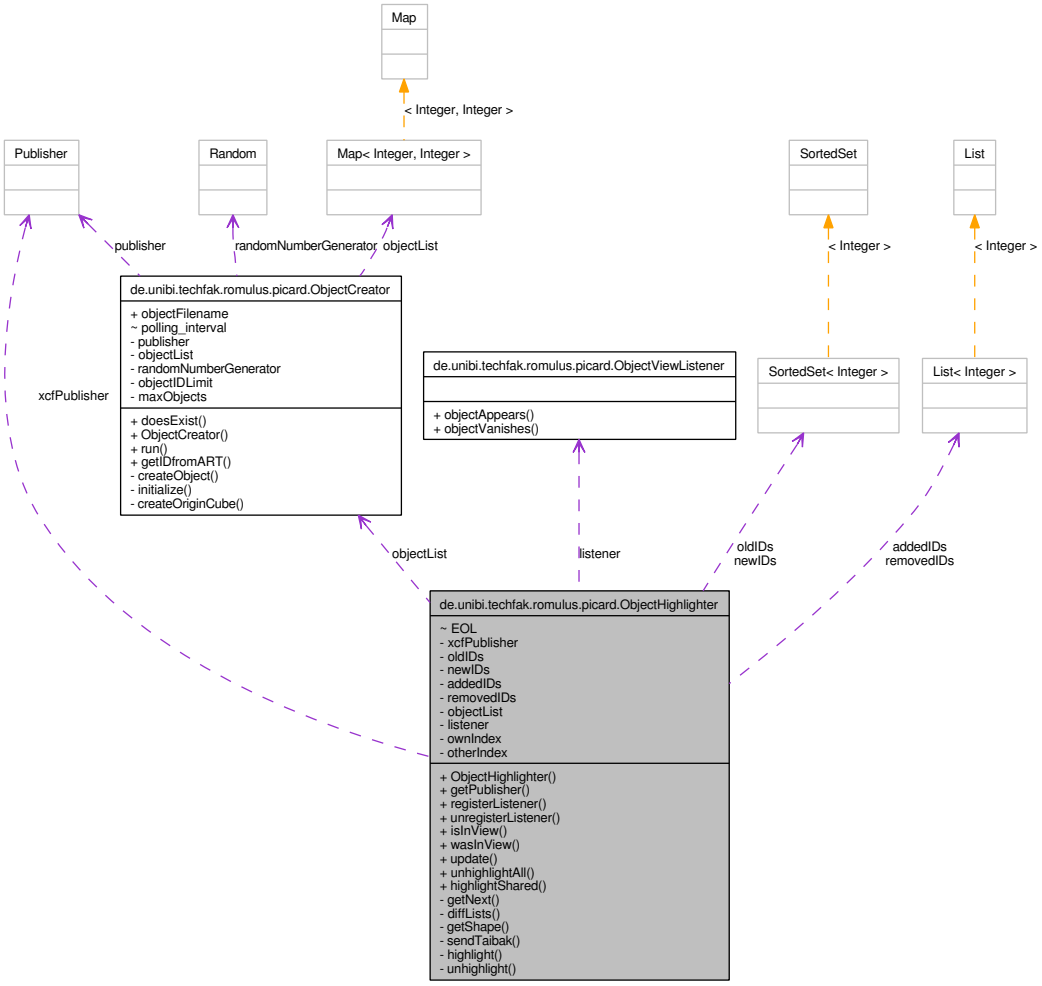
This variable contains the number of milliseconds to wait between checks for a remote subscriber listening to us.

The documentation for this class was generated from the following file:

- ObjectCreator.java

ObjectHighlighter Class Reference

Collaboration diagram for ObjectHighlighter:



Detailed Description

Gets the list of visible ART markers and thereby determines which objects to highlight and to unhighlight.

Author:

cmertes

Public Member Functions

- **ObjectHighlighter** (Publisher publisher, ObjectCreator creator, int index)
- Publisher **getPublisher** ()
- void registerListener (ObjectViewListener objectViewListener)

  Registers a single ObjectViewListener whose methods get called when an ART object appears or disappears.

- void unregisterListener ()

  The same as `registerListener(null)`.

- boolean isInView (Integer artID)

  Returns whether an ART marker with a given ID is currently in view.

- boolean wasInView (Integer artID)

  Returns whether an ART marker with a given ID was in view during the last iteration.

- void update (SortedSet< Integer > artIDs)

  Updates the list of ART IDs that are in view.

- void unhighlightAll ()

  Use this when object highlighting is switched from on to off to unhighlight the remaining highlighted objects.

- void highlightShared ()

  Use this when object highlighting is switched from off to on to highlight the currently unhighlighted objects in the partner's FOV.

Package Attributes

- final int EOL = -1

  End Of List.

Private Member Functions

- int getNext (Iterator< Integer > iter)

A very small helper function to diffLists().

- void diffLists ()
  Determines which objects are new and which disappeared.

- String getShape (Properties.envelope_t env)
  Associates a Properties.envelope_t with a string that can be used in taibaks.

- void sendTaibak (String highlightXML, Integer objectID)
  Sends a taibak to highlight or unhighlight a given object.

- void highlight (Integer objectID)
  Highlights an object.

- void unhighlight (Integer objectID)
  Unhighlights an object.

Private Attributes

- Publisher **xcfPublisher**
- SortedSet< Integer > **oldIDs**
- SortedSet< Integer > **newIDs**
- List< Integer > **addedIDs**
- List< Integer > **removedIDs**
- ObjectCreator **objectList**
- ObjectViewListener **listener** = null
- int **ownIndex**
- int **otherIndex**

Member Function Documentation

void ObjectHighlighter.registerListener (ObjectViewListener *objectViewListener*)

Registers a single ObjectViewListener whose methods get called when an ART object appears or disappears.

Currently only a single ObjectViewListener can be registered at any given time.

Parameters:

objectViewListener The ObjectViewListener which shall be informed of appearing and disappearing objects in the FOV of the current HMD

int ObjectHighlighter.getNext (Iterator< Integer > *iter*) `[private]`

A very small helper function to diffLists().

Returns the next element in a list of Integers or the `EOL` constant if there is no next element.

Parameters:

iter An iterator to a list of `Integers`

Returns:

The respective `int` (*not* `Integer`)

void ObjectHighlighter.diffLists () `[private]`

Determines which objects are new and which disappeared.

For this the function takes the current list C and the last list L of visible markers and generates the two lists `addedIDs` $A = C \setminus (L \cap C) = C \setminus L$ of markers that appeared since the last update from LAFORGE and `removedIDs` $R = L \setminus (L \cap C) = L \setminus C$ of markers that disappeared ($L \cap C$ terms added for descriptiveness).

This is done by iterating through both ascendingly sorted lists at the same time: When both IDs are equal (i.e. the current ID is $\in L \cap C$), both iterators are advanced. If the old ID is smaller, it is added to the list of disappeared objects and only this iterator is advanced, if the new ID is smaller, it is consequently added to the list of appeared objects and the new ID iterator is advanced. This is repeated until the end of both lists is reached.

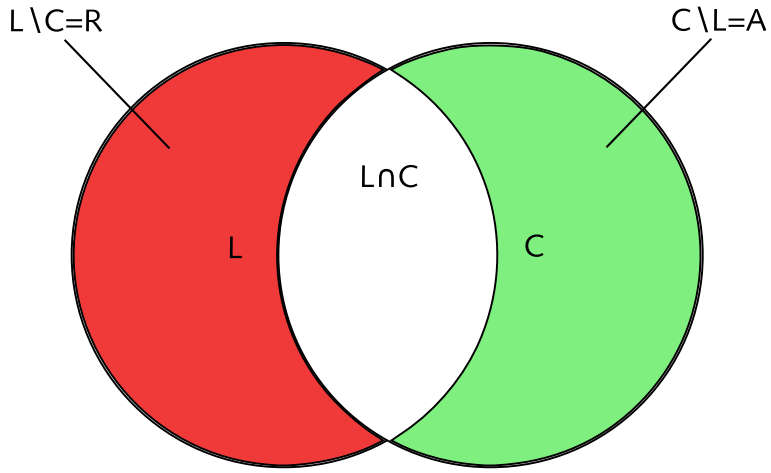


Figure 39: Venn diagram of the sets diffLists() uses

String ObjectHighlighter.getShape (Properties.envelope_t *env*)
`[private]`

Associates a Properties.envelope_t with a string that can be used in taibaks.

Parameters:

env The envelope type to translate

Returns:

The associated string

void ObjectHighlighter.sendTaibak (String *highlightXML*, Integer *objectID*) `[private]`

Sends a taibak to highlight or unhighlight a given object.

Parameters:

highlightXML A string containing the XML that constitutes the successors of the <OBJECT> tag.

objectID The object ID of the object to (un)highlight

void ObjectHighlighter.highlight (Integer *objectID*) `[private]`

Highlights an object.

Parameters:

objectID The object ID of the object to highlight

void ObjectHighlighter.unhighlight (Integer *objectID*) `[private]`

Unhighlights an object.

Parameters:

objectID The object ID of the object to unhighlight

boolean ObjectHighlighter.isInView (Integer *artID*)

Returns whether an ART marker with a given ID is currently in view.

Parameters:

artID The ID of the AR Toolkit marker to check for.

Returns:

`true` or `false`

boolean ObjectHighlighter.wasInView (Integer *artID*)

Returns whether an ART marker with a given ID was in view during the last iteration.

Parameters:

artID The ID of the AR Toolkit marker to check for.

Returns:

`true` or `false`

void ObjectHighlighter.update (SortedSet< Integer > *artIDs*)

Updates the list of ART IDs that are in view.

Compares this new list with the last one and figures out which objects are new and which can no longer be seen. Then it highlights and unhighlights accordingly and calls the appropriate methods of the registered ObjectViewListener (if any).

Parameters:

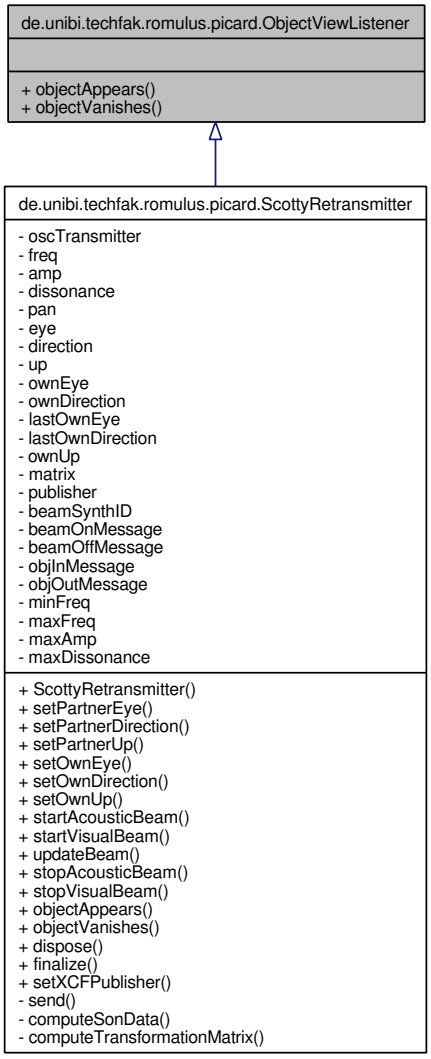
artIDs A list of ART IDs

The documentation for this class was generated from the following file:

- ObjectHighlighter.java

ObjectViewListener Interface Reference

Inheritance diagram for ObjectViewListener:



Detailed Description

An instance of a class inheriting from this interface can be registered to an ObjectHighlighter which will in turn notify the object when objects appear and disappear from the user's view.

Author:

cmertes

Public Member Functions

- void objectAppears ()

  This method will be called by ObjectHighlighter whenever an object is detected that wasn't there in the previous time step.

- void objectVanishes ()

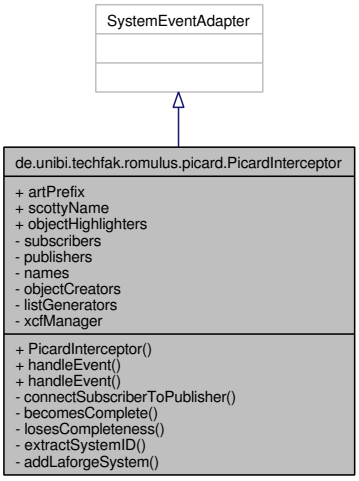
  This method will be called by ObjectHighlighter every time an object is no longer detected although it was there in the last time step.

The documentation for this interface was generated from the following file:

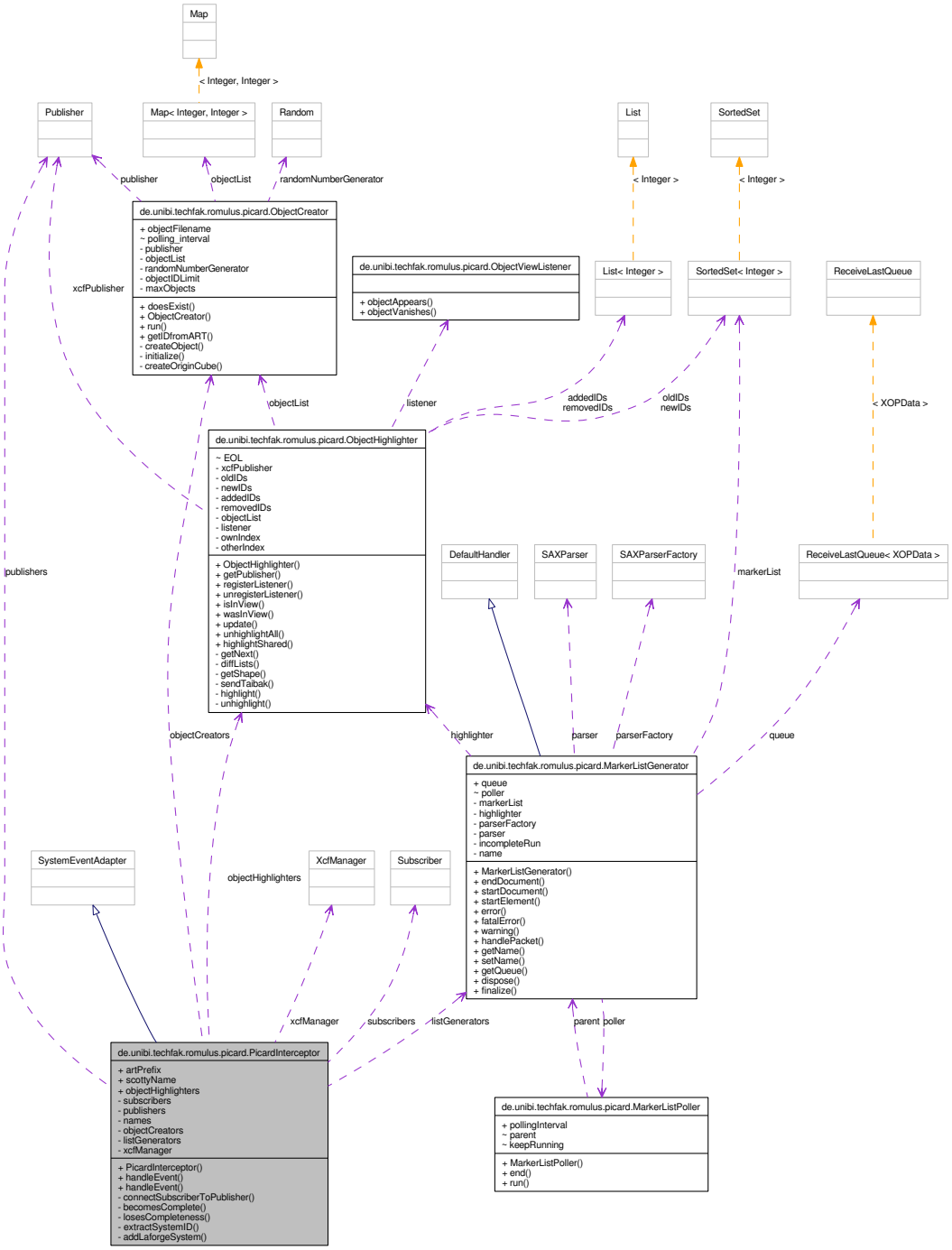
- ObjectViewListener.java

PicardInterceptor Class Reference

Inheritance diagram for PicardInterceptor:



Collaboration diagram for PicardInterceptor:



Detailed Description

This class monitors the XCF dispatcher and triggers actions according to relevant events.

This is the most central back-end class apart from Main. The latter instantiates this class directly after it takes over.

Author:

cmertes

Public Member Functions

- PicardInterceptor (XcfManager manager)

  The class constructor.

- void handleEvent (AdditionEvent e)

  Checks for new XCF entities whether they are LAFORGE or SCOTTY subscribers.

- void handleEvent (DeletionEvent e)

  Checks for newly disappeared XCF entities whether they were registered ones to clean up when necessary.

Public Attributes

- final String artPrefix = "ARTCOORDS"

  The prefix by which ART.

- final String scottyName = "SCOTTY"

  The name of the SCOTTY publisher.

- ObjectHighlighter[] **objectHighlighters** = {null, null}

Private Member Functions

- void connectSubscriberToPublisher (int id1, int id2)

As soon as both LAFORGE systems are present this function gets called twice, each time connect the output of one system to the input of the other.

- boolean becomesComplete ()

  Gets called as soon as both LAFORGE systems are present.

- void losesCompleteness ()

  If both LAFORGE systems were present but at least one ceases to exist (or to be visible for that matter), the actions of becomesComplete need to be undone which this function does.

- Integer extractSystemID (String publisherName)

  A little helper function to extract the system ID from an ARTCOORDS publisher name.

- boolean addLaforgeSystem (String publisherName)

  When a new LAFORGE system appears this method is called.

Private Attributes

- Subscriber[] **subscribers** = {null, null}
- Publisher[] **publishers** = {null, null}
- String[] **names** = {null, null}
- ObjectCreator[] **objectCreators** = {null, null}
- MarkerListGenerator[] **listGenerators** = {null, null}
- XcfManager **xcfManager**

Constructor & Destructor Documentation

PicardInterceptor.PicardInterceptor (XcfManager *manager*)

The class constructor.

Takes an XcfManager and uses it to check for already existing publishers (LAFORGE and SCOTTY).

Parameters:

manager The XcfManager object

Member Function Documentation

void PicardInterceptor.connectSubscriberToPublisher (int *id1*, int *id2*) `[private]`

As soon as both LAFORGE systems are present this function gets called twice, each time connect the output of one system to the input of the other.

This is crucial for all visualizations used.

Parameters:

id1 The system ID of the data producing LAFORGE system (output system)

id2 The system ID of the receiving LAFORGE system (input system)

boolean PicardInterceptor.becomesComplete () `[private]`

Gets called as soon as both LAFORGE systems are present.

Then both subscribers are created and connectSubscriberToPublisher(int, int) is called twice.

Returns:

`false` if the subscriber creation failed, `true` othewise

Integer PicardInterceptor.extractSystemID (String *publisherName*) `[private]`

A little helper function to extract the system ID from an ARTCOORDS publisher name.

Parameters:

publisherName ARTCOORDS_1 or ARTCOORDS_2

Returns:

0 or 1 (or `null` if no valid system ID could be extracted)

boolean PicardInterceptor.addLaforgeSystem (String *publisherName*)
`[private]`

When a new LAFORGE system appears this method is called.

It creates a publisher to send taibaks to this LAFORGE system and uses this via ObjectCreator to initialize it with ART objects. Finally it checks whether this is the second LAFORGE already to call becomesComplete() in this case.

Parameters:

publisherName The name of the new system's publisher

Returns:

`true` if everything went fine

void PicardInterceptor.handleEvent (AdditionEvent *e*)

Checks for new XCF entities whether they are LAFORGE or SCOTTY subscribers.

Gets called by the XcfManager.

void PicardInterceptor.handleEvent (DeletionEvent *e*)

Checks for newly disappeared XCF entities whether they were registered ones to clean up when necessary.

Gets called by the XcfManager.

Member Data Documentation

final String PicardInterceptor.artPrefix = "ARTCOORDS"

The prefix by which ART.

coordinate sending XCF publishers are identified.

The documentation for this class was generated from the following file:

- PicardInterceptor.java

Properties Class Reference

Detailed Description

In this class properties and settings are stored, set and read out again.

Command-line arguments as well as GUI elements manipulate the various parts of the program they affect via this class. It also contains some constants (a.k.a. *magic numbers*) that are needed at various places throughout the program.

Author:

cmertes

Public Types

- enum **envelope_t**

Public Member Functions

- void beRelative (boolean b)

  Sets the `relativeValues` *variable which determines, whether envelope durations are transformed to a parameter* α *(*`true`*) or not (*`false`*).*

- boolean isRelative ()

  Returns the `relativeValues` *variable.*

- void setAttackEnvelope (envelope_t env)

  Sets the shape of the highlight attack envelope.

- envelope_t getAttackEnvelope ()

  Gets the shape of the highlight attack envelope.

- void setDecayEnvelope (envelope_t env)

  Sets the shape of the highlight decay envelope.

- envelope_t getDecayEnvelope ()

  Gets the shape of the highlight decay envelope.

- void setAttackTime (double t)

  Sets the time of the attack phase of the highlight envelope.

- double getAttackValue ()

  Gets the α *parameter of the attack envelope.*

- double getAttackTime ()

  Gets the parameter setAttackTime was last called with.

- void setDecayTime (double t)

  Sets the time of the decay phase of the highlight envelope.

- double getDecayTime ()

  Gets the parameter setDecayTime was last called with.

- double getDecayValue ()

  Gets the α *parameter of the decay envelope.*

- String getDecayTerm ()

  Gets the attribute the <ENVELOPE> tag takes for the decay envelope settings.

- String getAttackTerm ()

  Gets the attribute the <ENVELOPE> tag takes for the attack envelope settings.

- boolean useSoundEvents ()

  Returns whether the event sonification is switched on.

- void useSoundEvents (boolean b)

  Sets whether the event sonification should be used.

- boolean useFoVSound ()

  Returns whether the FOV sonification is switched on.

- void useFoVSound (boolean b)

  Sets whether the FOV sonification should be used.

- boolean useFoVDisplay ()

  Returns whether the fuzzy rectangle FOV visualization is switched on.

- void useFoVDisplay (boolean b)

  Sets whether the fuzzy rectangle FOV visualization should be used.

- boolean useSonification ()

  Returns whether any sonification is switched on.

- boolean useScotty ()

  Returns whether any feature that uses SCOTTY is switched on.

- boolean useHighlighting ()

  Returns whether the object highlighting is turned on.

- void useHighlighting (boolean b)

  Sets whether the object highlighting is turned on.

- boolean useMemory ()

  Returns whether the object highlighting uses any attack or decay envelope.

- double getFuzzyRectWidth ()

  Gets the width of the rectangle to project onto the table surface to mark the user's FOV.

- double getFuzzyRectHeight ()

  Gets the height of the rectangle to project onto the table surface to mark the user's FOV.

- double getFuzzylength ()

  Gets the width of the fuzzy area around the rectangle which is projected onto the table surface to mark the user's FOV.

- void setFuzzylength (double d)

  Sets the amount of fuzzyness the FOV display has.

- void doCreateOriginCube (boolean b)

  Whether to create a unit cube in the origin of a new LAFORGE system.

- boolean doCreateOriginCube ()

  Whether to create a unit cube in the origin of a new LAFORGE system.

- boolean alwaysPing ()

Whether event sounds shall always be triggered or only when the event happens in the field of view of the user.

- void alwaysPing (boolean doAlwaysPing)

  Sets if event sounds shall always be triggered or only when the event happens in the field of view of the user.

Static Public Member Functions

- static envelope_t parseEnvelope (String envstr)

  The different shapes the attack and decay parts of the envelope can take.

- static double calcA (envelope_t env, double t)

  Given an envelope shape and a time this method calculates the parameter α *the <ENVELOPE> tag takes.*

Static Public Attributes

- static final double maxHighlight = 1.0

  The value an object is fully highlighted with.

- static final int controlRate = 100

  Frequency of LAFORGE's control rate.

- static final double expBase = 1.0

  Internal LAFORGE constant.

- static final double epsFactor = 1.0/5000

  Another internal LAFORGE constant.

- static final int fuzzyRectID = 0xFADE

  The fix object ID used in taibaks to refer to the fuzzy rectangle.

Private Attributes

- boolean **relativeValues** = true

- envelope_t **attackEnvelope** = envelope_t.ENV_EXP
- envelope_t **decayEnvelope** = envelope_t.ENV_LOG
- double **attackTime**
- double **attackValue**
- double **decayTime**
- double **decayValue**
- double **fuzzyRectWidth** = 376.0/515.0
- double **fuzzyRectHeight** = 48.0/103.0
- double **fuzzylength** = 3.0
- boolean **use_sound_events** = false
- boolean **use_fov_sound** = false
- boolean **use_fov_display** = false
- boolean **use_highlighting** = false
- boolean **create_origin_cube** = false
- boolean **alwaysPing** = true

Member Function Documentation

static envelope_t Properties.parseEnvelope (String *envstr*) `[static]`

The different shapes the attack and decay parts of the envelope can take.

Identifies an envelope type from a string.

Parameters:

envstr A string starting either with "log", "lin" or "exp"

Returns:

The according envelope_t value or `ENV_INVALID` if no match was found.

static double Properties.calcA (envelope_t *env*, double *t*) `[static]`

Given an envelope shape and a time this method calculates the parameter a the <ENVELOPE> tag takes.

Parameters:

env A member of envelope_t

t The time to convert in seconds

Returns:

The value that produces an attack or decay time of `t` or `NaN` if an invalid envelope type was passed. The latter should never happen.

void Properties.beRelative (boolean *b*)

Sets the `relativeValues` variable which determines, whether envelope durations are transformed to a parameter α (`true`) or not (`false`).

If this flag is *not* set, is has the effect that every time an envelope gets updated the timer restarts at 0 yet the value and thus the difference doesn't. This means the actual speed the value changes with varies, which is probably not what you want. Therefore the `relativeValues` flag is set by default. If the time within a value reaches 0 or its maximum is more important though than the speed it does this with, then you should call this function with `false`.

Parameters:

b Set `true` for absolute value change speed and relative value change time, set `false` for the opposite.

boolean Properties.isRelative ()

Returns the `relativeValues` variable.

See beRelative(boolean) for an explanation.

Returns:

`true` or `false`, see above

void Properties.setAttackEnvelope (envelope_t *env*)

Sets the shape of the highlight attack envelope.

Parameters:

env A value from envelope_t

envelope_t Properties.getAttackEnvelope ()

Gets the shape of the highlight attack envelope.

Returns:

A value from envelope_t

void Properties.setDecayEnvelope (envelope_t *env*)

Sets the shape of the highlight decay envelope.

Parameters:

env A value from envelope_t

envelope_t Properties.getDecayEnvelope ()

Gets the shape of the highlight decay envelope.

Returns:

A value from envelope_t

void Properties.setAttackTime (double *t*)

Sets the time of the attack phase of the highlight envelope.

Parameters:

t The attack time in seconds

double Properties.getAttackValue ()

Gets the α parameter of the attack envelope.

Returns:

The internal `attackValue` variable

double Properties.getAttackTime ()

Gets the parameter setAttackTime was last called with.

Returns:

The internal `attackTime` variable

void Properties.setDecayTime (double *t*)

Sets the time of the decay phase of the highlight envelope.

Parameters:

t The decay time in seconds

double Properties.getDecayTime ()

Gets the parameter setDecayTime was last called with.

Returns:

The internal `decayTime` variable

double Properties.getDecayValue ()

Gets the α parameter of the decay envelope.

Returns:

The internal `decayValue` variable

String Properties.getDecayTerm ()

Gets the attribute the <ENVELOPE> tag takes for the decay envelope settings.

Returns:

a="*`decayValue`*" or `t="`*`decayTime`*" (with *`decayValue`* and *`decayTime`* replaced by the according values of course), depending on whether or not `relativeValues` is set.

String Properties.getAttackTerm ()

Gets the attribute the <ENVELOPE> tag takes for the attack envelope settings.

Returns:

a="*`attackValue`*" or `t="`*`attackTime`*" (with *`attackValue`* and *`attackTime`* replaced by the according values of course), depending on whether or not `relativeValues` is set.

boolean Properties.useSoundEvents ()

Returns whether the event sonification is switched on.

Returns:

`true` if the event sonification is switched on, `false` otherwise.

void Properties.useSoundEvents (boolean *b*)

Sets whether the event sonification should be used.

Parameters:

b Set `true` if the event sonification shall be switched on or `false` otherwise.

boolean Properties.useFoVSound ()

Returns whether the FOV sonification is switched on.

Returns:

`true` if the FOV sonification is switched on, `false` otherwise.

void Properties.useFoVSound (boolean *b*)

Sets whether the FOV sonification should be used.

Parameters:

b Set `true` if the FOV sonification shall be switched on or `false` otherwise.

boolean Properties.useFoVDisplay ()

Returns whether the fuzzy rectangle FOV visualization is switched on.

Returns:

`true` if the continuous FOV visualization is switched on, `false` otherwise.

void Properties.useFoVDisplay (boolean *b*)

Sets whether the fuzzy rectangle FOV visualization should be used.

Parameters:

b Set `true` if the FOV visualization shall be switched on or `false` otherwise.

boolean Properties.useSonification ()

Returns whether any sonification is switched on.

Returns:

`true` if either the event sonification or the FOV sonification is switched on or both, `false` otherwise.

boolean Properties.useScotty ()

Returns whether any feature that uses SCOTTY is switched on.

Returns:

`true` if either the FOV sonification or the FOV visualization or both are switched on, `false` otherwise.

boolean Properties.useHighlighting ()

Returns whether the object highlighting is turned on.

Returns:

`true` if the object highlighting visualization is switched on, `false` otherwise.

void Properties.useHighlighting (boolean *b*)

Sets whether the object highlighting is turned on.

Parameters:

b `true` if the object highlighting visualization shall be switched on, `false` otherwise.

See also:

setAttackTime(double)
setDecayTime(double)
setAttackEnvelope(envelope_t)
setDecayEnvelope(envelope_t)

boolean Properties.useMemory ()

Returns whether the object highlighting uses any attack or decay envelope.

Although just an attack phase isn't what one would normally call "memory" but well, in the context of this function it is.

Returns:

`true` if at least one of `attackTime` and `decayTime` is non-zero, `false` if both are.

double Properties.getFuzzyRectWidth ()

Gets the width of the rectangle to project onto the table surface to mark the user's FOV.

Doesn't include the fuzzy area, whose extent is read by getFuzzylength().

Returns:

The rectangle's extent along the x axis.

double Properties.getFuzzyRectHeight ()

Gets the height of the rectangle to project onto the table surface to mark the user's FOV.

Doesn't include the fuzzy area, whose extent is read by getFuzzylength().

Returns:

The rectangle's extent along the y axis.

double Properties.getFuzzylength ()

Gets the width of the fuzzy area around the rectangle which is projected onto the table surface to mark the user's FOV.

There's no difference between the extent along the x and the y axis.

Returns:

The amount of fuzzyness the FOV display has.

void Properties.setFuzzylength (double *d*)

Sets the amount of fuzzyness the FOV display has.

While the actual FOV is always solid, there is a colour gradient around it whose length is set with this method.

Parameters:

d A non-negative value.

void Properties.doCreateOriginCube (boolean *b*)

Whether to create a unit cube in the origin of a new LAFORGE system.

Use for testing purposes. Defaults to `false`

Parameters:

b Set to `true` to automatically create a unit cube around the origin of every registered LAFORGE system.

boolean Properties.doCreateOriginCube ()

Whether to create a unit cube in the origin of a new LAFORGE system.

See also:

doCreateOriginCube(boolean)

Returns:

The value of the internal `create_origin_cube` variable.

boolean Properties.alwaysPing ()

Whether event sounds shall always be triggered or only when the event happens in the field of view of the user.

Returns:

`true` when all events trigger sounds, `false` when only visible events trigger sounds.

void Properties.alwaysPing (boolean *doAlwaysPing*)

Sets if event sounds shall always be triggered or only when the event happens in the field of view of the user.

Parameters:

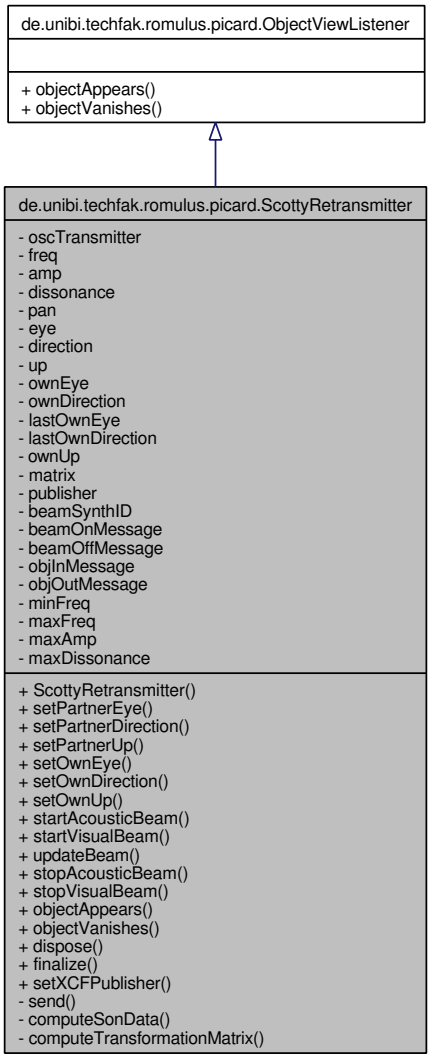
doAlwaysPing `true` to make every event trigger a sound, `false` to make only visible events trigger a sound.

The documentation for this class was generated from the following file:

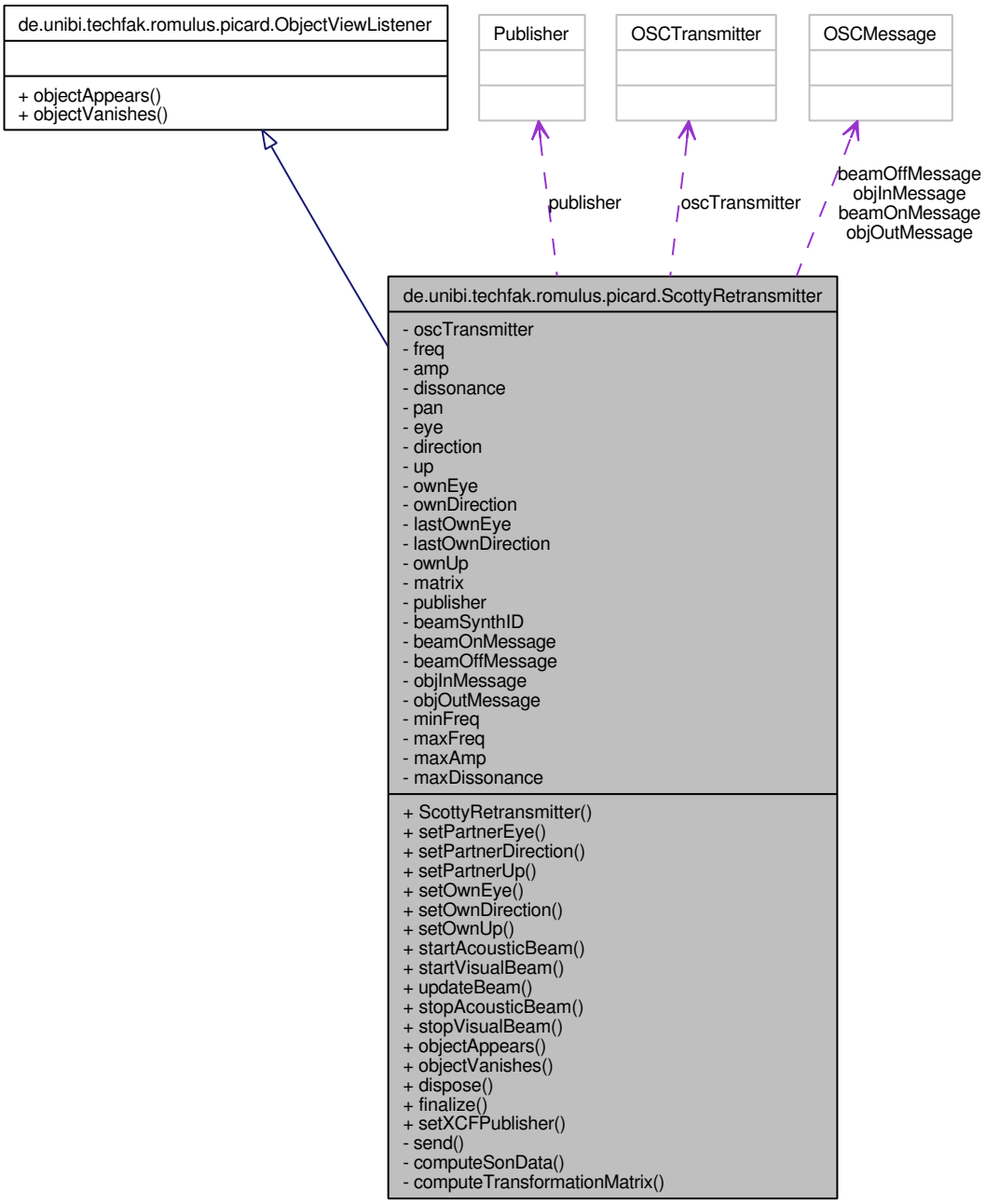
– Properties.java

ScottyRetransmitter Class Reference

Inheritance diagram for ScottyRetransmitter:



Collaboration diagram for ScottyRetransmitter:



Detailed Description

Gets information about a user's gaze direction, transforms it into OSC packets and sends these to a given network socket.

This socket is typically a host running `scsynth`. This class also uses the same gaze direction data to create a fuzzy rectangle via XCF. Thirdly it implements ObjectViewListener to send event sound OSC packages. This and Sonificator were only one class if Java supported multiple inheritance.

Author:

cmertes

Public Member Functions

- ScottyRetransmitter (InetSocketAddress socket)

  The class constructor.

- void setPartnerEye (double[] vec)

  This method updates the direction vector of the user's dialogue partner's gaze.

- void setPartnerDirection (double[] vec)

  This method updates the direction vector of the user's dialogue partner's gaze.

- void setPartnerUp (double[] vec)

  This method updates the up vector of the user's dialogue partner's gaze.

- void setOwnEye (double[] vec)

  This method updates the direction vector of the user's own gaze.

- void setOwnDirection (double[] vec)

  This method updates the direction vector of the user's own gaze.

- void setOwnUp (double[] vec)

  This method updates the up vector of the user's own gaze.

- boolean startAcousticBeam ()

  Starts the continuous sonification of the gaze of the user's dialogue partner.

- void startVisualBeam ()

  Starts the fuzzy rectangle augmentation.

- void updateBeam ()

  Updates the continuous sonification of the gaze of the user's dialogue partner with the most recent data.

- boolean stopAcousticBeam ()

  Stops the continuous sonification of the gaze of the user's dialogue partner.

- void stopVisualBeam ()

  Stops the fuzzy rectangle augmentation.

- void objectAppears ()

  Sends an event packet to the network socket that is associated with the sound of an appearing object.

- void objectVanishes ()

  Sends an event packet to the network socket that is associated with the sound of an disappearing object.

- void dispose ()

  Switches off all active augmentations and closes the OSC network channel.

- void **finalize** ()
- void setXCFPublisher (Publisher publisher)

  Sets the XCF publisher that is used by this class to send taibaks to LAFORGE.

Private Member Functions

- boolean **send** (OSCMessage message)
- void computeSonData ()

  This method uses the gaze direction data of both interlocutors to compute the sonification raw data and maps these to the sonification parameters.

- void computeTransformationMatrix ()

  Calculates the transformation matrix $\mathbf{M}$.

Private Attributes

- OSCTransmitter **oscTransmitter**
- Integer **freq**
- Double **amp**
- Double **dissonance**
- Double **pan**
- double[] **eye**
- double[] **direction**
- double[] **up**
- double[] **ownEye**
- double[] **ownDirection**
- double[] **lastOwnEye**
- double[] **lastOwnDirection**
- double[] **ownUp**
- double[][] **matrix** = null
- Publisher publisher = null

  The XCF publisher used to send SCOTTY-dependent taibaks to LAFORGE.

Static Private Attributes

- static final Integer **beamSynthID** = new Integer(100)
- static final OSCMessage **beamOnMessage**
- static final OSCMessage **beamOffMessage**
- static final OSCMessage **objInMessage**
- static final OSCMessage **objOutMessage**
- static final int **minFreq** = 50
- static final int **maxFreq** = 1000
- static final double **maxAmp** = 5.0
- static final double **maxDissonance** = 10.0

Constructor & Destructor Documentation

ScottyRetransmitter.ScottyRetransmitter (InetSocketAddress *socket*)

The class constructor.

There needs to be one objects for every LAFORGE system.

Parameters:

socket The data structure containing target host name and port number.

Member Function Documentation

void ScottyRetransmitter.setPartnerEye (double[] *vec*)

This method updates the direction vector of the user's dialogue partner's gaze.

This data is normally provided by SCOTTY.

Parameters:

vec A three-dimensional direction vector.

See also:

updateBeam()

void ScottyRetransmitter.setPartnerDirection (double[] *vec*)

This method updates the direction vector of the user's dialogue partner's gaze.

This data is normally provided by SCOTTY.

Parameters:

vec A three-dimensional direction vector.

See also:

updateBeam()

void ScottyRetransmitter.setPartnerUp (double[] *vec*)

This method updates the up vector of the user's dialogue partner's gaze.

This data is normally provided by SCOTTY.

Parameters:

vec A three-dimensional up vector.

See also:

updateBeam()

void ScottyRetransmitter.setOwnEye (double[] *vec*)

This method updates the direction vector of the user's own gaze.

This data is normally provided by SCOTTY.

Parameters:

vec A three-dimensional direction vector.

See also:

updateBeam()

void ScottyRetransmitter.setOwnDirection (double[] *vec*)

This method updates the direction vector of the user's own gaze.

This data is normally provided by SCOTTY.

Parameters:

vec A three-dimensional direction vector.

See also:

updateBeam()

void ScottyRetransmitter.setOwnUp (double[] *vec*)

This method updates the up vector of the user's own gaze.

This data is normally provided by SCOTTY.

Parameters:

vec A three-dimensional up vector.

See also:

updateBeam()

boolean ScottyRetransmitter.startAcousticBeam ()

Starts the continuous sonification of the gaze of the user's dialogue partner.

Returns:

`true` if the OSC packet was sent successfully, `false` otherwise. Please note that this doesn't eye the packet could be received successfully by the remote end.

See also:

updateBeam()
stopAcousticBeam()

void ScottyRetransmitter.updateBeam ()

Updates the continuous sonification of the gaze of the user's dialogue partner with the most recent data.

Does nothing if Main.sonificator.useBeam is not set. Calling this method before startAcousticBeam() will lead to an error on the remote end.

Returns:

`true` if the OSC packet was sent successfully, `false` otherwise. Please note that this doesn't eye the packet could be received successfully

by the remote end. If Main.sonificator.useBeam is not set, `false` is returned.

See also:

startAcousticBeam()
stopAcousticBeam()

void ScottyRetransmitter.computeTransformationMatrix () [private]

Calculates the transformation matrix $\mathbf{M}$.

$$\mathbf{M} = \begin{pmatrix} \texttt{eye[2]} & 0 & -\texttt{eye[0]} & 0 \\ 0 & \texttt{eye[2]} & -\texttt{eye[1]} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & \texttt{eye[2]} \end{pmatrix} \begin{pmatrix} \texttt{s[0]} & \texttt{s[1]} & \texttt{s[2]} & 0 \\ \texttt{v[0]} & \texttt{v[1]} & \texttt{v[2]} & 0 \\ -\texttt{direction[0]} & -\texttt{direction[1]} & -\texttt{direction[2]} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & \texttt{eye[0]} \\ 0 & 1 & 0 & \texttt{eye[1]} \\ 0 & 0 & 1 & \texttt{eye[2]} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

boolean ScottyRetransmitter.stopAcousticBeam ()

Stops the continuous sonification of the gaze of the user's dialogue partner.

This only makes sense when startAcousticBeam() had been called before.

Returns:

`true` if the OSC packet was sent successfully, `false` otherwise. Please note that this doesn't eye the packet could be received successfully by the remote end.

See also:

startAcousticBeam()
updateBeam()

void ScottyRetransmitter.objectAppears ()

Sends an event packet to the network socket that is associated with the sound of an appearing object.

No call to an initialization function is required.

See also:

objectVanishes()
ObjectViewListener

Implements ObjectViewListener.

void ScottyRetransmitter.objectVanishes ()

Sends an event packet to the network socket that is associated with the sound of an disappearing object.

No call to an initialization function is required.

See also:

objectAppears()
ObjectViewListener

Implements ObjectViewListener.

void ScottyRetransmitter.setXCFPublisher (Publisher *publisher*)

Sets the XCF publisher that is used by this class to send taibaks to LAFORGE.

This should be the same publisher that is used for the other taibaks so we can't create it ourselves here.

Parameters:

publisher Either the publisher that should be used or `null` to delete the publisher reference in the current ScottyRetransmitter instance

Member Data Documentation

final OSCMessage ScottyRetransmitter.beamOnMessage `[static, private]`

Initial value:

```
new OSCMessage("/s_new", new Object[] {
                        "fovSon", beamSynthID, new Integer(0), new Integer(0)
        })
```

final OSCMessage ScottyRetransmitter.beamOffMessage [static, private]

Initial value:

```
new OSCMessage("/n_free",
                        new Object[] {beamSynthID})
```

final OSCMessage ScottyRetransmitter.objInMessage [static, private]

Initial value:

```
new OSCMessage("/s_new", new Object[] {
                        "objIn", new Integer(-1), new Integer(0), new Integer(0)
        })
```

final OSCMessage ScottyRetransmitter.objOutMessage [static, private]

Initial value:

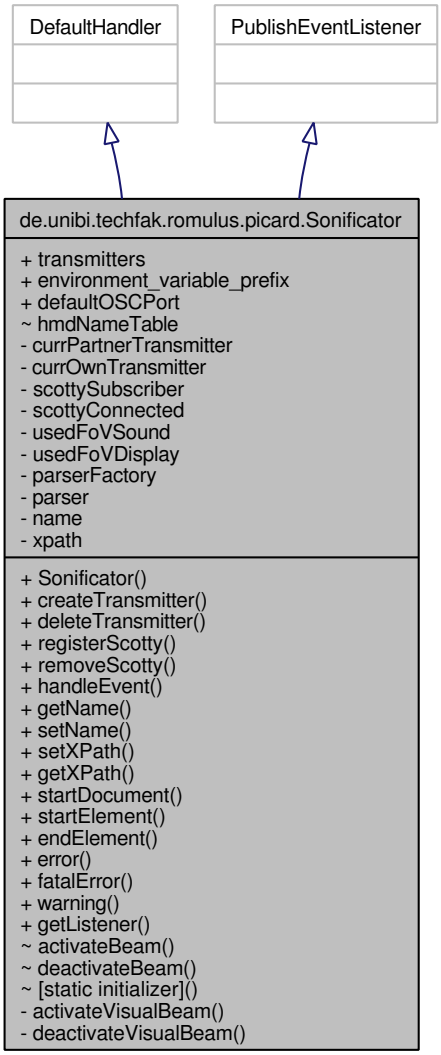
```
new OSCMessage("/s_new", new Object[] {
                        "objOut", new Integer(-1), new Integer(0), new Integer(0)
        })
```

The documentation for this class was generated from the following file:

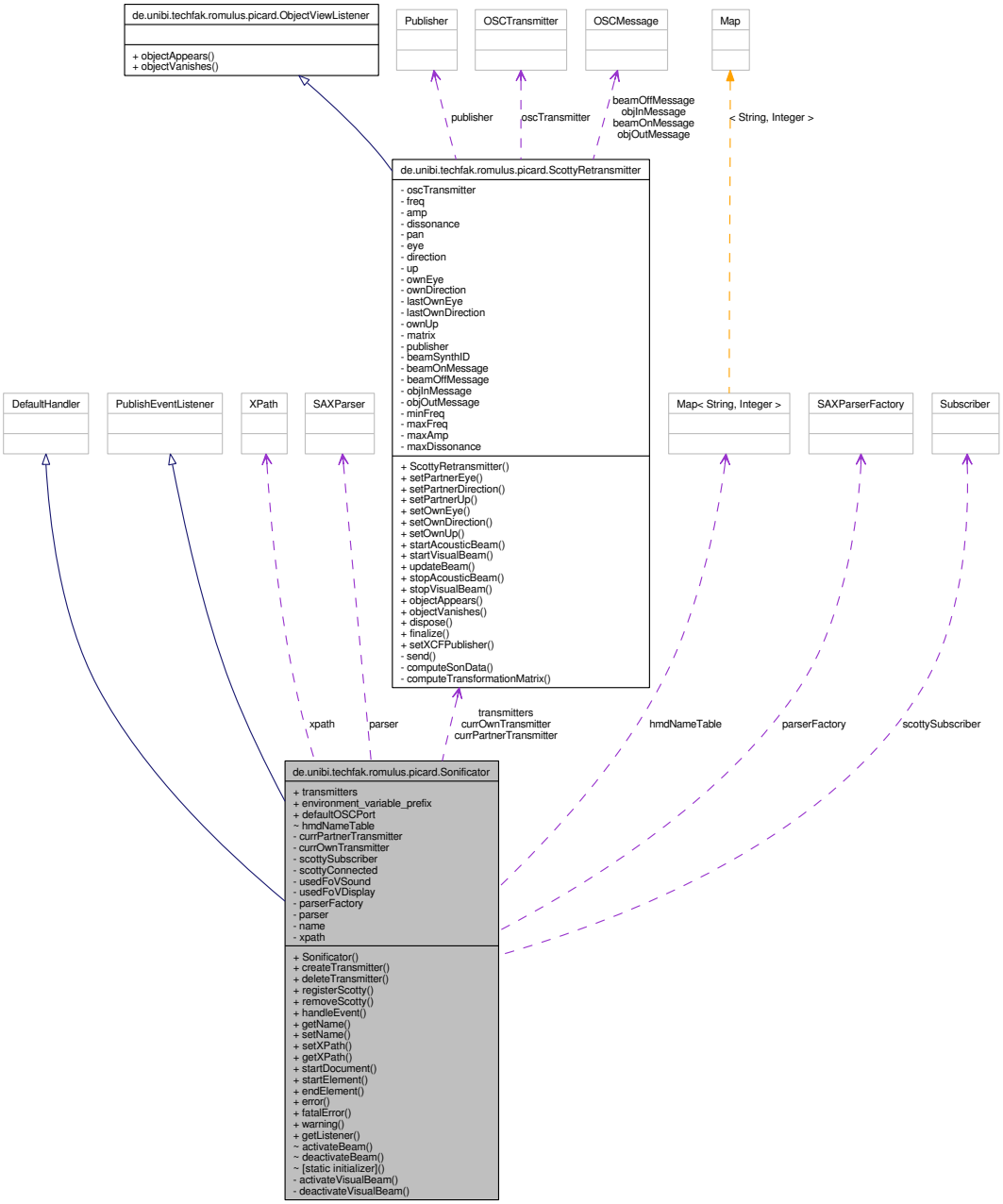
- ScottyRetransmitter.java

Sonificator Class Reference

Inheritance diagram for Sonificator:



Collaboration diagram for Sonificator:



Detailed Description

This class listens to the SCOTTY data and triggers sonifications *and* – which admittedly makes the name a bit misleading – field-of-view visualization accordingly.

It attends to the other sonifications as well, perhaps justifying the name a bit. This and ScottyRetransmitter were only one class if Java supported multiple inheritance.

Author:

cmertes

Public Member Functions

- Sonificator ()

  Class constructor.

- boolean createTransmitter (int systemID)

  Creates a new ScottyRetransmitter.

- void deleteTransmitter (int systemID)

  Deactivates and removes the specified ScottyRetransmitter.

- boolean registerScotty (String scottyPublisherName)

  Creates an XCF subscriber to SCOTTY given the name of its publisher.

- void removeScotty ()

  Deactivates and deletes the SCOTTY subscriber and so stops receiving SCOTTY data.

- void handleEvent (PublishEvent e)

  Gets called by the XCF subscriber when a SCOTTY data packet comes in.

- String **getName** ()
- void **setName** (String s)
- void **setXPath** (XPath xp)
- XPath **getXPath** ()
- void startDocument ()

Initializes data extraction from SCOTTY packets.

- void startElement (String namespaceURI, String localName, String qName, Attributes atts)

  Reads the actual vectors from the SCOTTY XML data.

- void endElement (String uri, String localName, String qName)

  Updates the appropriate ScottyRetransmitter with the new data just received from SCOTTY.

- void **error** (SAXParseException exception)
- void **fatalError** (SAXParseException exception)
- void **warning** (SAXParseException exception)
- ObjectViewListener getListener (int index)

  Grants access to the two ScottyRetransmitters.

Public Attributes

- ScottyRetransmitter[] transmitters = {null, null}

  Array of ScottyRetransmitters.

Static Public Attributes

- static final String environment_variable_prefix = "LAFORGE_SC_-HOST"

  What the environment variable containing the OSC host/port data begins with.

- static final int defaultOSCPort = 57110

  Which port to send OSC packets to if not otherwise specified by the environment variable.

Package Functions

- boolean activateBeam ()

  Activates both FOV sonifications.

– void deactivateBeam ()

Deactivates both FOV sonifications.

Static Package Functions

– **[static initializer]**

Static Package Attributes

– static Map< String, Integer > hmdNameTable

Hard-coded map which associates HMD names with their according system IDs.

Private Member Functions

– boolean activateVisualBeam ()

Activates FOV visualizations for both HMDs.

– void deactivateVisualBeam ()

Deactivates FOV vizualizations for both HMDs.

Private Attributes

– ScottyRetransmitter currPartnerTransmitter = null

The transmitter for which the current SCOTTY HMD is the partner's.

– ScottyRetransmitter currOwnTransmitter = null

The transmitter for which the current SCOTTY HMD is its own.

– Subscriber scottySubscriber = null

The XCF subscriber listening to SCOTTY.

– boolean scottyConnected = false

`true` *if we are currently subscribed to SCOTTY*

- boolean usedFoVSound = false

  Whether props.useFoVSound() was true when the last SCOTTY packet arrived.

- boolean usedFoVDisplay = false

  Whether props.useFoVDisplay() was true when the last SCOTTY packet arrived.

- SAXParserFactory **parserFactory**
- SAXParser parser

  SAX parser used to parse SCOTTY packets using this class as the event handler.

- String **name** = "SCOTTY_LISTENER"
- XPath **xpath** = new XPath("/∗")

Constructor & Destructor Documentation

Sonificator.Sonificator ()

Class constructor.

Only creates the SAX parser. The rest is done when according methods are called.

Member Function Documentation

boolean Sonificator.createTransmitter (int *systemID*)

Creates a new ScottyRetransmitter.

Parameters:

systemID The system ID which the transmitter should send its OSC packets to. Must be 1 or 2, other values lead to undefined behaviour.

Returns:

`true` if the transmitter was correctly created, `false` otherwise.

void Sonificator.deleteTransmitter (int *systemID*)

Deactivates and removes the specified ScottyRetransmitter.

Parameters:

systemID Either 1 or 2

boolean Sonificator.activateBeam () `[package]`

Activates both FOV sonifications.

For a non-existent transmitter a warning is emitted.

Returns:

Currently always `true`

void Sonificator.deactivateBeam () `[package]`

Deactivates both FOV sonifications.

For a non-existent transmitter a warning is emitted.

boolean Sonificator.activateVisualBeam () `[private]`

Activates FOV visualizations for both HMDs.

For a non-existent transmitter a warning is emitted.

Returns:

Currently always `true`

void Sonificator.deactivateVisualBeam () `[private]`

Deactivates FOV vizualizations for both HMDs.

For a non-existent transmitter a warning is emitted.

boolean Sonificator.registerScotty (String *scottyPublisherName*)

Creates an XCF subscriber to SCOTTY given the name of its publisher.

Also starts listening to this publisher and thus parsing its packets as soon as they arrive.

Parameters:

scottyPublisherName The name of the SCOTTY publisher to subscribe to

Returns:

`false` if some error occured, `true` otherwise

void Sonificator.handleEvent (PublishEvent *e*)

Gets called by the XCF subscriber when a SCOTTY data packet comes in.

Triggers its parsing.

void Sonificator.startDocument ()

Initializes data extraction from SCOTTY packets.

Gets called by the SAX parser.

void Sonificator.startElement (String *namespaceURI*, String *localName*, String *qName*, Attributes *atts*)

Reads the actual vectors from the SCOTTY XML data.

Gets called by the SAX parser.

void Sonificator.endElement (String *uri*, String *localName*, String *qName*)

Updates the appropriate ScottyRetransmitter with the new data just received from SCOTTY.

Gets called by the SAX parser.

ObjectViewListener Sonificator.getListener (int *index*)

Grants access to the two ScottyRetransmitters.

Parameters:

index Which ScottyRetransmitter to return: 0 or 1

Returns:

The ScottyRetransmitter object requested (might be `null` if `createTransmitter(index)` was not called before)

The documentation for this class was generated from the following file:

– Sonificator.java

C.2 LAFORGE Class Documentation

ARTObjectList Class Reference

`#include <globallists.h>`

Detailed Description

The list of those objects bound to ART markers.

See also:

ObjectList

Public Types

- typedef ObjectList::ObjectPtr **ObjectPtr**

Public Member Functions

- void **conditional_draw** (ICLDrawWidget3D *, const graphObject::artID) const
- void **clear** ()
- bool **does_exist** (const graphObject::artID) const
- bool **add** (const graphObject::artID, ObjectPtr &)
- bool **remove** (const graphObject::artID id)
- graphObject::artID **get_artID** (const ObjectPtr &obj)
- graphObject * **get_object** (const graphObject::artID)
- void **release_object** () const

Private Types

- typedef std::map< graphObject::artID, ObjectPtr > **map_t**
- typedef std::map< graphObject::artID, ObjectPtr >::iterator **iter_t**

Private Member Functions

- void **lock** () const
- void **unlock** () const

Static Private Attributes

- static map_t **art_object_list**
- static icl::Mutex **art_object_list_mutex**

Friends

- class ObjectList

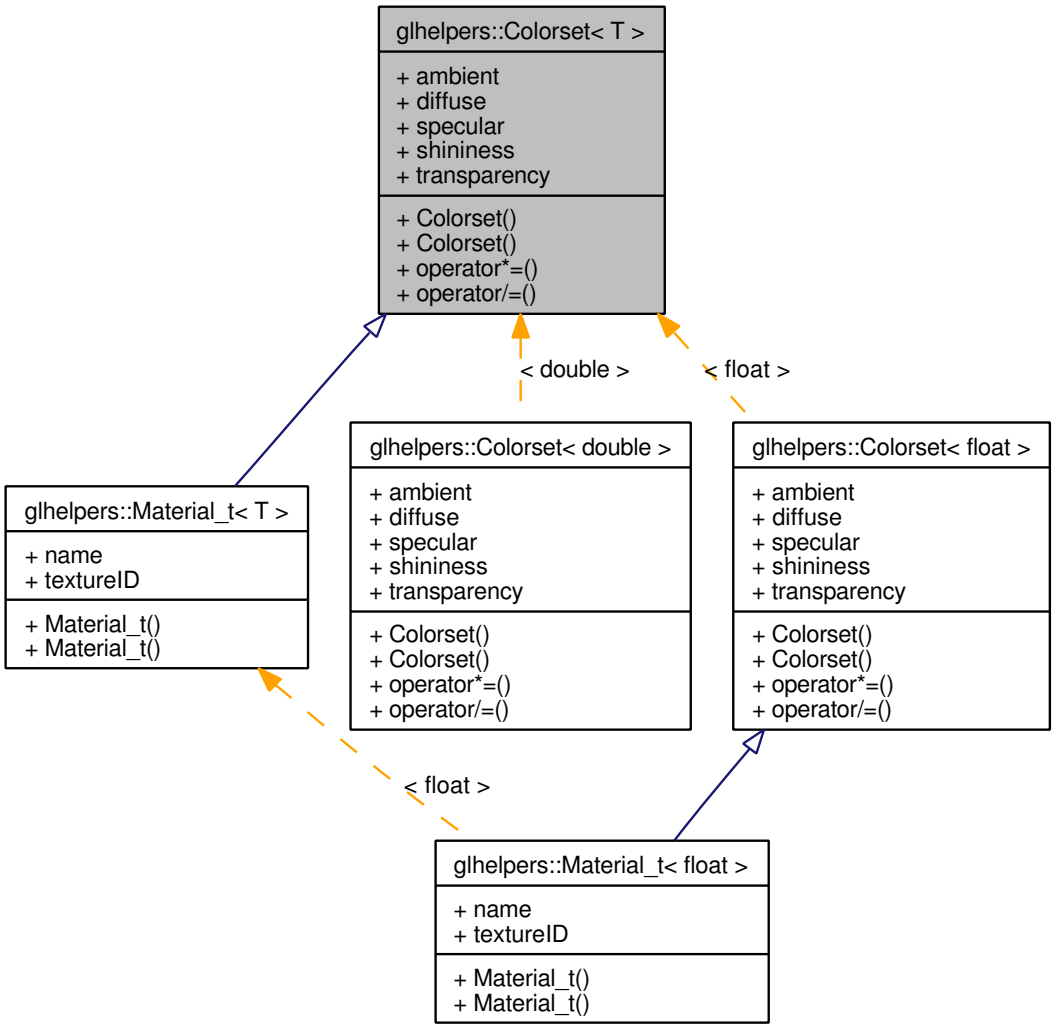
The documentation for this class was generated from the following files:

- globallists.h
- globallists.cpp

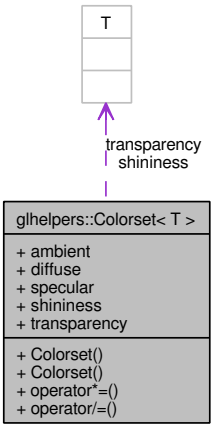
glhelpers::Colorset< T > Class Template Reference

`#include <glhelpers.h>`

Inheritance diagram for glhelpers::Colorset< T >:



Collaboration diagram for glhelpers::Colorset< T >:



Detailed Description

template<class T = float> class glhelpers::Colorset< T >

This data structure stores all colour values OpenGL uses.

Public Member Functions

- **Colorset** (const Colorset &cs)
- Colorset< T > & **operator*=** (const Colorset< T > &rhs)
- Colorset< T > & **operator/=** (T rhs)

Public Attributes

- Vector3< T > **ambient**
- Vector3< T > **diffuse**
- Vector3< T > **specular**
- T **shininess**
- T **transparency**

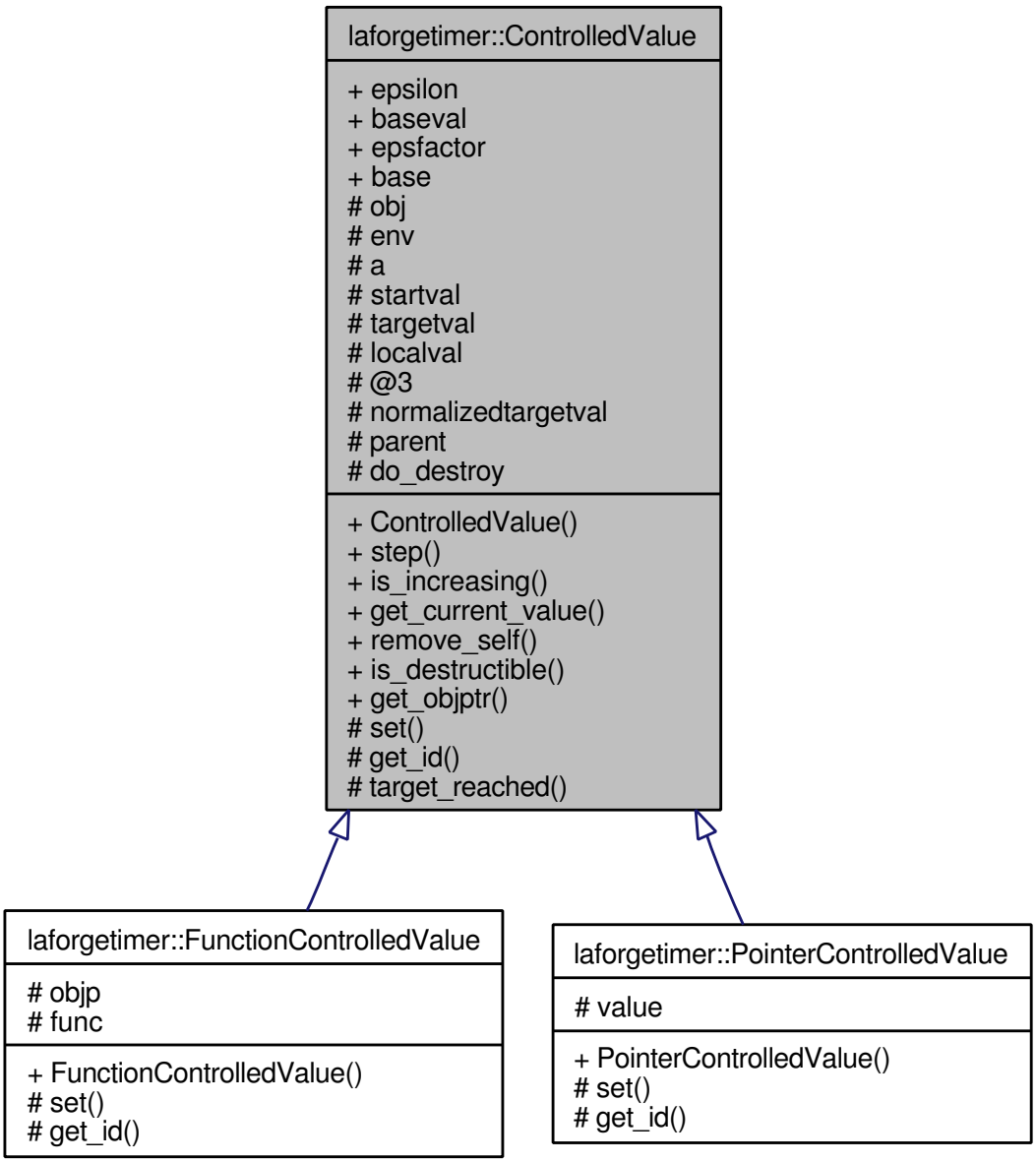
The documentation for this class was generated from the following file:

– glhelpers.h

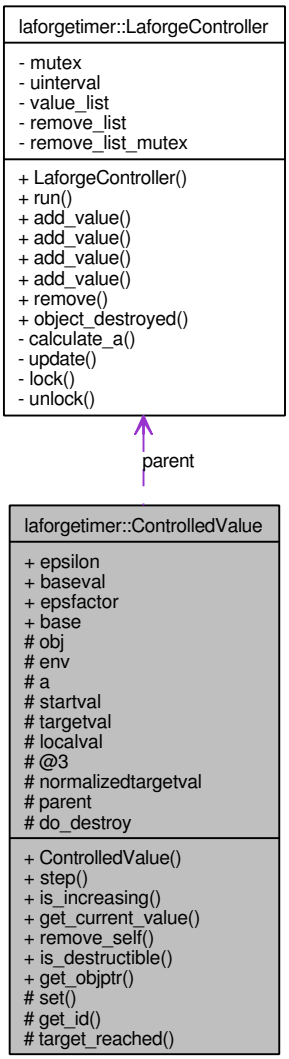
laforgetimer::ControlledValue Class Reference

`#include <laforgetimer.h>`

Inheritance diagram for laforgetimer::ControlledValue:



Collaboration diagram for laforgetimer::ControlledValue:



Detailed Description

A value that is controlled by LaforgeTimer so that it changes automatically.

This can be used to realize fade-in and fade-out like effects. *Do not instantiate these classes yourself!* Use LaforgeTimer.

Public Member Functions

- **ControlledValue** (LaforgeController *parent_lc, ObjectList::ObjectPtr objptr, double start, double target, double stepsize, envelope_t envelope, bool destructible)
- void **step** () const
- bool **is_increasing** () const
- double **get_current_value** () const
- void **remove_self** () const
- bool **is_destructible** () const
- ObjectList::ObjectPtr **get_objptr** () const

Static Public Attributes

- static const double **epsfactor** = 1.0/5000
- static const double **base** = 1.0

Protected Member Functions

- virtual void **set** (double val) const =0
- virtual ValueID **get_id** () const =0
- void **target_reached** () const

Protected Attributes

- ObjectList::ObjectPtr **obj**
- envelope_t **env**
- double **a**
- double **startval**
- double **targetval**
- double **localval**
- union {
  double **epsilon**
  double **baseval**
  };
- double **normalizedtargetval**
- LaforgeController * **parent**

- bool **do_destroy**

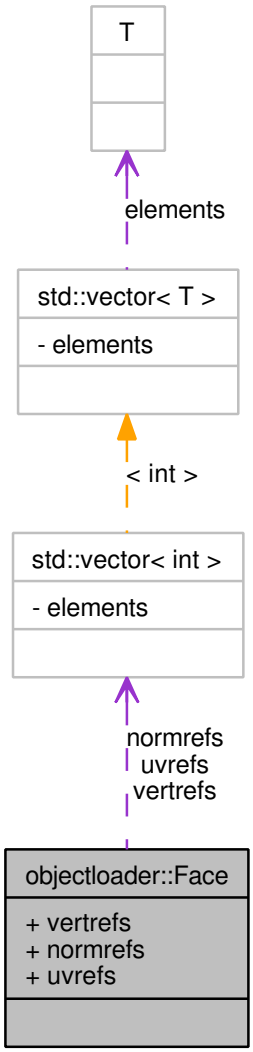
The documentation for this class was generated from the following files:

- laforgetimer.h
- laforgetimer.cpp

objectloader::Face Struct Reference

`#include <objectloader.h>`

Collaboration diagram for objectloader::Face:



Detailed Description

A single face of an OpenGL mesh object.

Public Attributes

- vector< int > **vertrefs**
- vector< int > **normrefs**
- vector< int > **uvrefs**

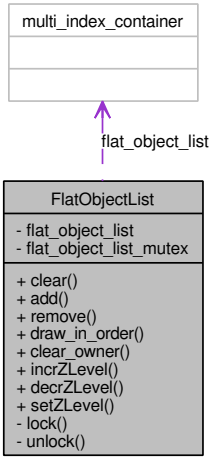
The documentation for this struct was generated from the following file:

- objectloader.h

FlatObjectList Class Reference

`#include <globallists.h>`

Collaboration diagram for FlatObjectList:



Detailed Description

The list of the 2D objects, sorted by their drawing order.

The underlying data structure is rather complicated and adding and removing objects - while being $O(\log n)$ - would be faster using a different approach. We do it this way though because we assume that creating and deleting objects are rare events compared to drawing objects. Should it ever turn out that objects are created and destroyed at a very high rate, it would perhaps be worth reconsidering using temporary lists recreated in every iteration of the render loop.

See also:

ObjectList

Public Types

– typedef ObjectList::ObjectPtr **ObjectPtr**

Public Member Functions

- void **clear** ()
- bool **add** (ObjectPtr &obj, XCFThread ∗owner, const int zLevel=0)
- void **remove** (ObjectPtr &obj)
- void **draw_in_order** (ICLDrawWidget3D ∗) const
- void **clear_owner** (XCFThread ∗owner)
- void **incrZLevel** (ObjectPtr &obj, int deltaz=1)
- void **decrZLevel** (ObjectPtr &obj, int deltaz=1)
- void **setZLevel** (ObjectPtr &obj, int z)

Private Types

- typedef multi_index_container< object_triple, indexed_by< ordered_non_unique< tag< creator >, member< object_triple, XCFThread ∗,&object_triple::owner > >, ordered_non_unique< tag< zlevel >, member< object_triple, int,&object_triple::zLevel > >, ordered_unique< tag< pointer >, const_mem_fun< object_triple, icl::ICLDrawWidget3D::GLCallback ∗,&object_triple::getObjPtr > > > > **collection_t**
- typedef collection_t::index< creator >::type **owner_t**
- typedef collection_t::index< zlevel >::type **zlevel_t**
- typedef collection_t::index< pointer >::type **object_t**
- typedef owner_t::iterator **owner_iter_t**
- typedef zlevel_t::iterator **zlevel_iter_t**
- typedef object_t::iterator **object_iter_t**

Private Member Functions

- void **lock** () const
- void **unlock** () const

Static Private Attributes

- static collection_t **flat_object_list**
- static icl::Mutex **flat_object_list_mutex**

Friends

- class ObjectList

Classes

- struct **creator**
- struct **object_triple**
- struct **pointer**
- struct **zlevel**

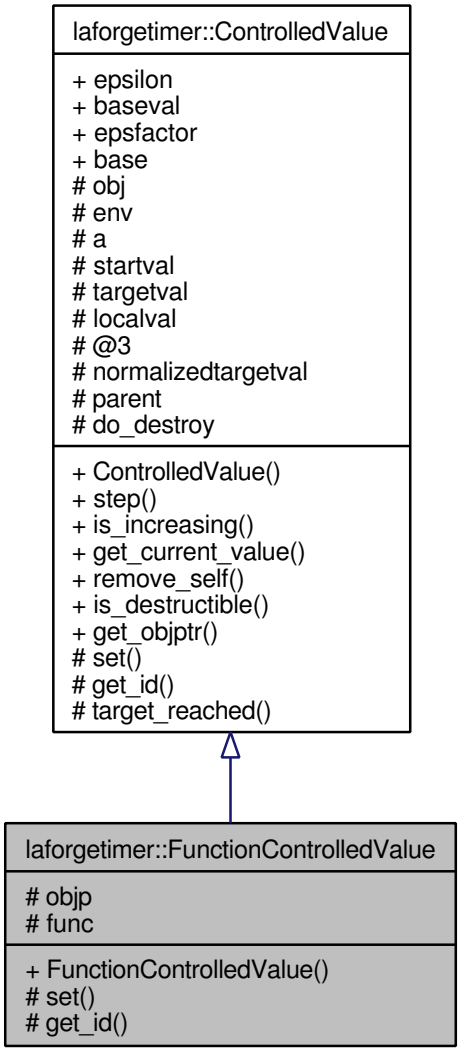
The documentation for this class was generated from the following files:

- globallists.h
- globallists.cpp

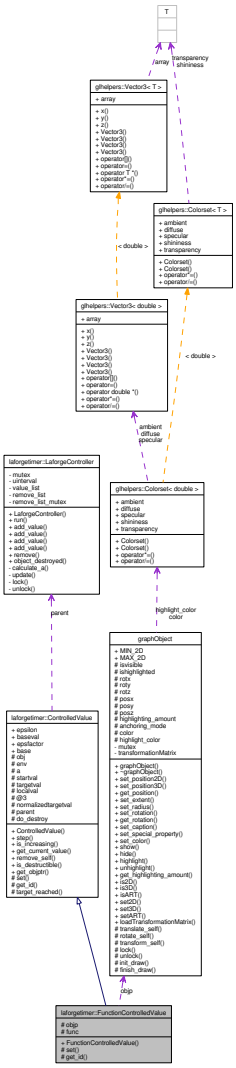
laforgetimer::FunctionControlledValue Class Reference

`#include <laforgetimer.h>`

Inheritance diagram for laforgetimer::FunctionControlledValue:



Collaboration diagram for laforgetimer::FunctionControlledValue:



Detailed Description

This is a ControlledValue that controls a value indirectly via a setter method or even more common via a wrapper around a setted method.

You need a value_function_t pointer to use this class. *Do not, however, instantiate it directly!*

Public Member Functions

- **FunctionControlledValue** (LaforgeController *parent_lc, ObjectList::ObjectPtr objptr, value_function_t function, double start, double target, double stepsize, envelope_t envelope, bool destructible)

Protected Member Functions

- virtual void **set** (double val) const
- virtual ValueID **get_id** () const

Protected Attributes

- graphObject * **objp**
- value_function_t **func**

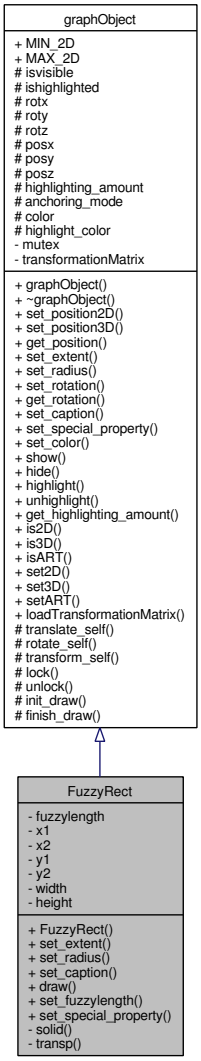
The documentation for this class was generated from the following file:

- laforgetimer.h

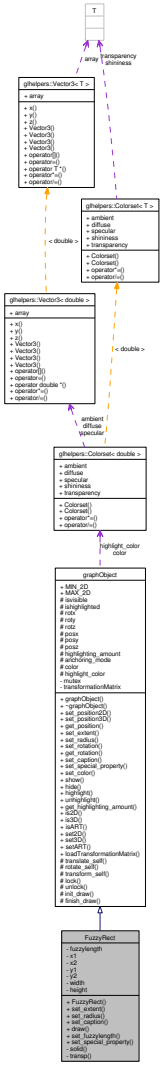
FuzzyRect Class Reference

`#include <FuzzyRect.h>`

Inheritance diagram for FuzzyRect:



Collaboration diagram for FuzzyRect:



Detailed Description

Draws a solid rectangle with width and height specified by set_extent and a rectangular frame around it that fades from solid to transparent.

The frame width is set via set_fuzzylength.

Author:

cmertes

Public Member Functions

- FuzzyRect ()

  The class constructor does nothing more than to assure that a `FuzzyRect` *is a 3D object per default so no position has to be specified to activate it being drawn.*

- virtual int set_extent (const int value, const char dimension= 'x')

  Sets the width or height of the rectangle.

- virtual int **set_radius** (const int value, const int num=1)
- virtual int **set_caption** (const string content)
- virtual void draw ()

  Draws the rectangle and the fuzzy area around it with the parameters set by set_extent(const int, const char) and (const double).

- void set_fuzzylength (const double fuzzy)

  Sets the width of the fuzzy area around the actual rectangle.

- virtual int set_special_property (const double value, const string name)

  Use this method to set the width of the fuzzy area.

Private Member Functions

- void solid ()

  Makes the following vertices opaque.

- void transp ()

  Makes the following vertices transparent.

Private Attributes

- double **fuzzylength**

- double **x1**
- double **x2**
- double **y1**
- double **y2**
- double **width**
- double **height**

Member Function Documentation

int FuzzyRect::set_extent (const int *value*, const char *dimension* = 'x') `[virtual]`

Sets the width or height of the rectangle.

These values don't include the fuzzy area around the rectangle. Use set_-fuzzylength(const double) for this.

Parameters:

value The value to set the current dimension to.

dimension `'x'` to set the width or `'y'` to set the height

Returns:

`true` if you specified a legal dimension, `false` otherwise

Implements graphObject.

void FuzzyRect::set_fuzzylength (const double *fuzzy*) `[inline]`

Sets the width of the fuzzy area around the actual rectangle.

You generally won't want to use this function directly as it is not part of the graphObject interface but FuzzyRect::set_special_property(const double, const string) instead.

Parameters:

fuzzy The value to set the length to.

Here is the caller graph for this function:

virtual int FuzzyRect::set_special_property (const double *value*, const string *name*) `[inline, virtual]`

Use this method to set the width of the fuzzy area.

Parameters:

value the value to set the length to

name must be "fuzzyness"

Returns:

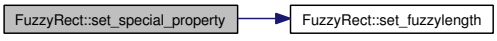
`true`. If the return value is `false` you passed the wrong string or you tried to set the fuzzylength of a graphObject that is no FuzzyRect.

See also:

graphObject::set_special_property(const double, const string)

Reimplemented from graphObject.

Here is the call graph for this function:



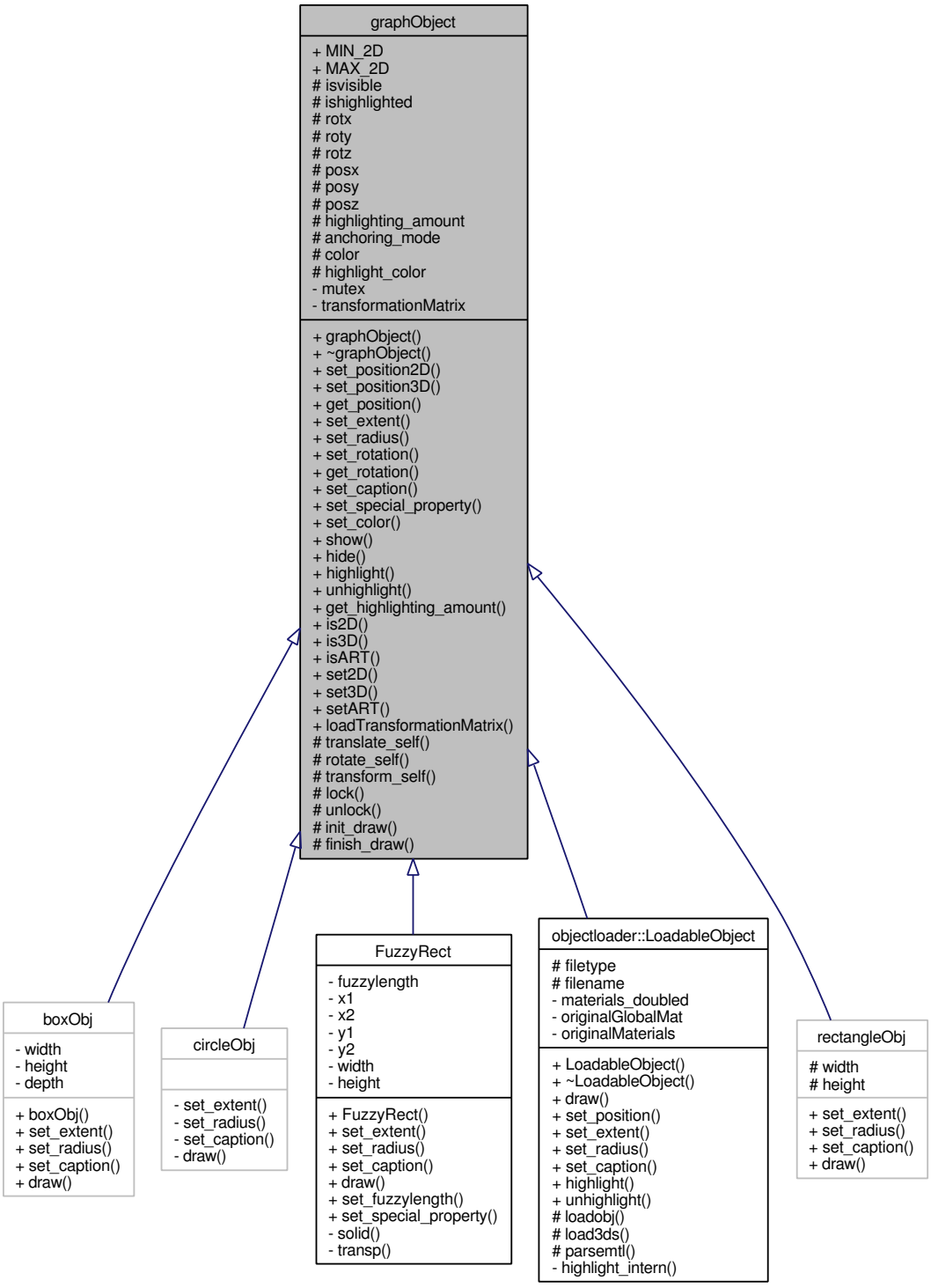
The documentation for this class was generated from the following files:

- FuzzyRect.h
- FuzzyRect.cpp

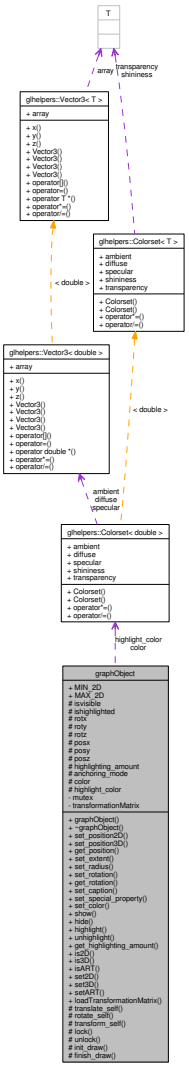
graphObject Class Reference

```
#include <graphobj.h>
```

Inheritance diagram for graphObject:



Collaboration diagram for graphObject:



Detailed Description

An abstract base class all graphical objects have to inherit from.

Author:

cmertes

Public Types

- enum **color_t** {

  COLOR_AMBIENT = 0, **COLOR_DIFFUSE** = 1, **COLOR_SPECULAR** = 2, **COLOR_SHININESS** = 3,

  HIGHLIGHT_AMBIENT = 4, **HIGHLIGHT_DIFFUSE** = 5, **HIGHLIGHT_SPECULAR** = 6, **HIGHLIGHT_SHININESS** = 7 }
- enum **anchor_t** { **ANCHOR_NOPOS**, **ANCHOR_3D**, **ANCHOR_2D**, **ANCHOR_ART** }
- typedef int **artID**

Public Member Functions

- virtual int **set_position2D** (const double value, const char dimension= 'x')
- virtual int **set_position3D** (const double value, const char dimension= 'x')
- virtual double **get_position** (const char dimension= 'x') const
- virtual int **set_extent** (const int value, const char dimension= 'x')=0
- virtual int **set_radius** (const int value, const int num=1)=0
- virtual int **set_rotation** (const double value, const char dimension= 'x')
- virtual double **get_rotation** (const char dimension= 'x') const
- virtual int **set_caption** (const string content)=0
- virtual int set_special_property (const double value, const string name)

  This function can be used to set properties only very few types of graphObjects have.

- virtual void **set_color** (double r, double *g*=-1.0, double b=-1.0, double a=-1.0, color_t color_type=COLOR_DIFFUSE)
- virtual void **show** ()
- virtual void **hide** ()
- virtual bool **highlight** (double amount=1.0)
- virtual bool **unhighlight** ()
- virtual double **get_highlighting_amount** ()
- bool **is2D** ()
- bool **is3D** ()
- bool **isART** ()
- void **set2D** ()

- void **set3D** ()
- void **setART** ()
- void loadTransformationMatrix (const vector< double > *const matrix)

  Loads an OpenGL transformation matrix that is applied before every drawing operation.

Static Public Attributes

- static const double **MIN_2D** = -1.0
- static const double **MAX_2D** = 1.0

Protected Member Functions

- void **translate_self** () const
- void **rotate_self** () const
- void **transform_self** () const
- void **lock** () const
- void **unlock** () const
- void **init_draw** () const
- void **finish_draw** () const

Protected Attributes

- bool **isvisible**
- bool **ishighlighted**
- double **rotx**
- double **roty**
- double **rotz**
- double **posx**
- double **posy**
- double **posz**
- double **highlighting_amount**
- anchor_t **anchoring_mode**
- Colorset< double > **color**

Static Protected Attributes

- static Colorset< double > **highlight_color** = glhelpers::Colorset<double>()

Private Attributes

- icl::Mutex **mutex**
- shared_ptr< vector< double > > **transformationMatrix**

Member Function Documentation

int graphObject::set_special_property (const double *value*, const string *name*) `[virtual]`

This function can be used to set properties only very few types of graphObjects have.

If they'd use their own methods for this, a run-time type checking would have to be done. This is circumvented with this general method for the small cost of a string comparison for every call to this function.

Parameters:

value the value the property is to be set to

name a string that should identify the property uniquely

Returns:

`true` only if the specified property is valid for the current object type and this property could be set correctly. `false` in every other case.

Reimplemented in FuzzyRect.

void graphObject::loadTransformationMatrix (const vector< double > *const *matrix*)

Loads an OpenGL transformation matrix that is applied before every drawing operation.

The matrix has the usual OpenGL form

$$\mathbf{A} = \begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{pmatrix}$$

with a[n] written as a_n for the sake of better readability. It will get multiplied to the current modelview matrix $\mathbf{M}$ such that $\mathbf{M}_{\text{new}} = \mathbf{MA}$.

Parameters:

matrix A pointer to the matrix or `null` to delete the current matrix. The matrix is copied internally so you can pass temporary objects or modify the data afterwards.

The documentation for this class was generated from the following files:

- graphobj.h
- graphobj.cpp

laforgetimer::LaforgeController Class Reference

`#include <laforgetimer.h>`

Detailed Description

An object of this class constitutes a thread that handles a list of ControlledValues.

Public Member Functions

- LaforgeController (unsigned long interval=10000)

  The class constructor takes the time between `update()`*s in microseconds as its argument.*

- void run ()

  This function is automatically run when the thread is started.

- bool **add_value** (ObjectList::ObjectPtr obj, double *value, double target_value, double a, envelope_t envelope=ENV_LIN, bool destructible=true)
- bool **add_value** (ObjectList::ObjectPtr obj, value_function_t func, double startvalue, double target_value, double a, envelope_t envelope=ENV_LIN, bool destructible=true)
- bool **add_value** (ObjectList::ObjectPtr obj, double *value, double target_value, unsigned usecs, envelope_t envelope=ENV_LIN, bool destructible=true)
- bool **add_value** (ObjectList::ObjectPtr obj, value_function_t func, double start_value, double target_value, unsigned usecs, envelope_t envelope=ENV_LIN, bool destructible=true)
- void **remove** (const ValueID &id)
- bool **object_destroyed** (ObjectList::ObjectPtr obj)

Private Member Functions

- double **calculate_a** (envelope_t env, unsigned t, double start, double target)
- void **update** ()

- void **lock** ()
- void **unlock** ()

Private Attributes

- icl::Mutex **mutex**
- const unsigned long **uinterval**
- map< ValueID, shared_ptr< ControlledValue > > **value_list**
- list< ValueID > **remove_list**
- icl::Mutex **remove_list_mutex**

Constructor & Destructor Documentation

laforgetimer::LaforgeController::LaforgeController (unsigned long *interval* = `10000`)

The class constructor takes the time between `update()`s in microseconds as its argument.

Parameters:

interval Time in s between one call to update() and the next.

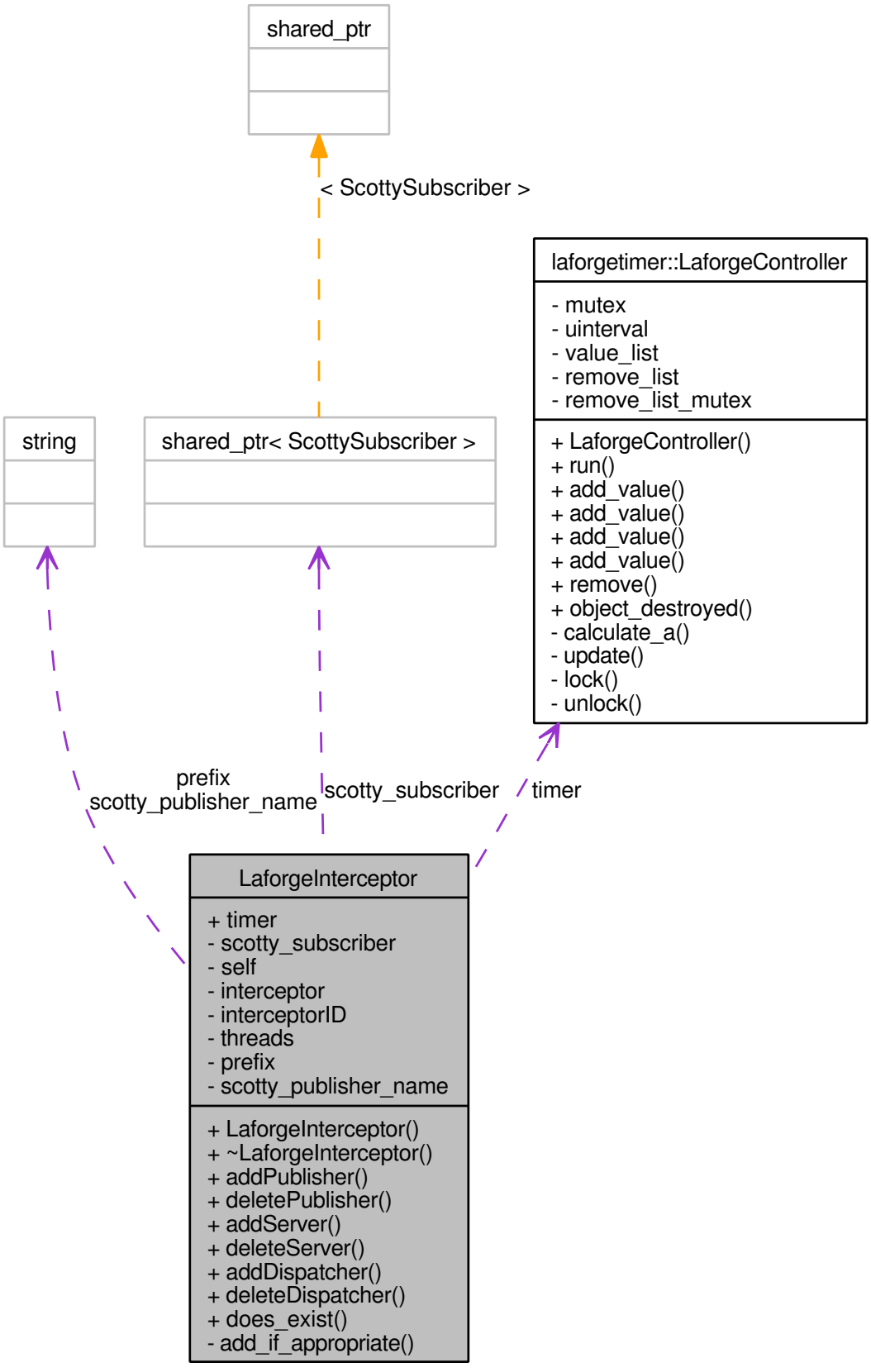
The documentation for this class was generated from the following files:

- laforgetimer.h
- laforgetimer.cpp

LaforgeInterceptor Class Reference

`#include <laforgeinterceptor.h>`

Collaboration diagram for LaforgeInterceptor:



Detailed Description

An instance of this class waits for new XCF publishers to appear.

When this happens, it checks if the publisher's name starts with a certain prefix and if this is the case a new XCFThread is created for this publisher.

Public Member Functions

- virtual void addPublisher (const string &name, const Remote::PublisherPrx &, const Ice::Current &=Ice::Current()) throw ()

  This function inherited from the DispatcherInterceptor interface is called whenever a new XCF publisher registers itself at the dispatcher.

- virtual void deletePublisher (const string &name, const Ice::Current &=Ice::Current()) throw ()

  This function implements the DispatcherInterceptor interface and is automatically called when an XCF publisher is deleted from the dispatcher.

- virtual void **addServer** (const std::string &, const Remote::ServerPrx &, const Ice::Current &=Ice::Current()) throw ()
- virtual void **deleteServer** (const std::string &, const Ice::Current &=Ice::Current()) throw ()
- virtual void **addDispatcher** (const std::string &, const Remote::DispatcherInfoPrx &, const Ice::Current &=Ice::Current()) throw ()
- virtual void **deleteDispatcher** (const std::string &, const Ice::Current &=Ice::Current()) throw ()
- bool does_exist (const std::string &publisherName)

  Checks whether a publisher certain name is already registered to LAFORGE.

Public Attributes

- laforgetimer::LaforgeController **timer**

Private Member Functions

- void **add_if_appropriate** (const string &publisherName)

Private Attributes

- shared_ptr< ScottySubscriber > **scotty_subscriber**
- DispatcherInterceptorPtr **self**
- XCFInterceptorPtr **interceptor**
- Ice::Identity **interceptorID**
- list< shared_ptr< XCFThread > > **threads**
- string **prefix**

Static Private Attributes

- static const string **scotty_publisher_name** = "SCOTTY"

Member Function Documentation

virtual void LaforgeInterceptor::addPublisher (const string & *name*, const Remote::PublisherPrx &, const Ice::Current & = `Ice::Current()`**) throw ()** `[inline, virtual]`

This function inherited from the DispatcherInterceptor interface is called whenever a new XCF publisher registers itself at the dispatcher.

See also:

deletePublisher(const string&, const Ice::Current&) throw()

virtual void LaforgeInterceptor::deletePublisher (const string & *name*, const Ice::Current & = `Ice::Current()`**) throw ()** `[inline, virtual]`

This function implements the DispatcherInterceptor interface and is automatically called when an XCF publisher is deleted from the dispatcher.

See also:

addPublisher(const string&, const Remote:PublisherPrx&, const Ice::Current&) throw()

bool LaforgeInterceptor::does_exist (const std::string & *publisherName*)
`[inline]`

Checks whether a publisher certain name is already registered to LAFORGE.

Parameters:

publisherName The publisher name to check for.

Returns:

`true` when a publisher with this name already exists, `false` otherwise.

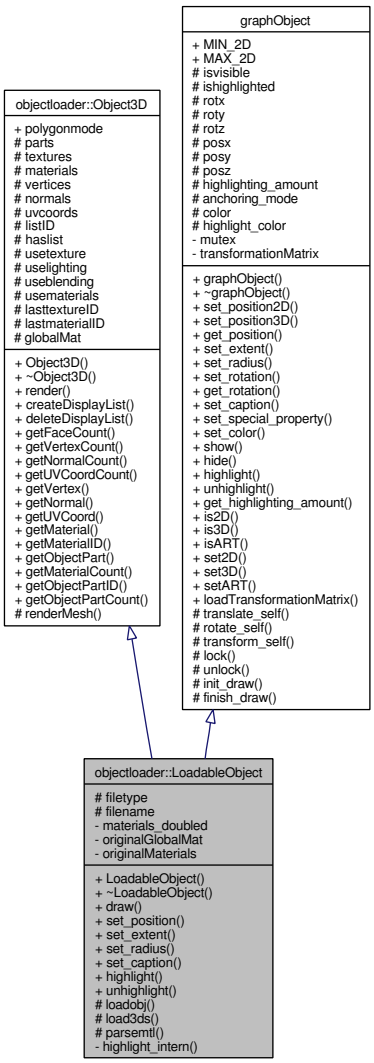
The documentation for this class was generated from the following files:

- laforgeinterceptor.h
- laforgeinterceptor.cpp

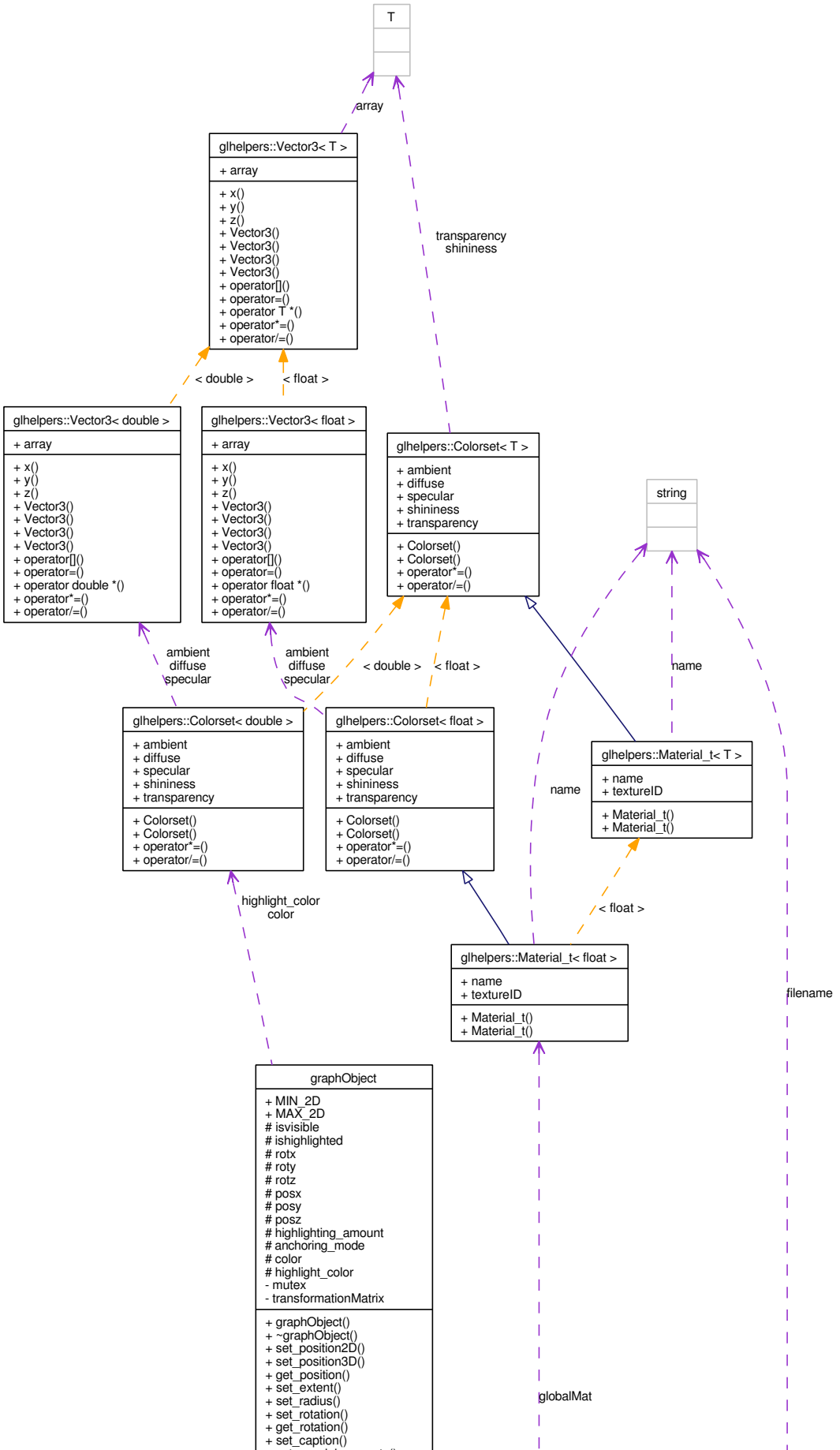
objectloader::LoadableObject Class Reference

`#include <objectloader.h>`

Inheritance diagram for objectloader::LoadableObject:



Collaboration diagram for objectloader::LoadableObject:



Detailed Description

This is a 3D OpenGL object that receives its data from a file containing the vertex data.

Material data can optionally be stored, too. At the moment only Wavefront .obj is supported.

Public Member Functions

- **LoadableObject** (const string &uri)
- virtual void **draw** ()
- virtual int **set_position** (const int value, const char dimension= 'x')
- virtual int **set_extent** (const int value, const char dimension= 'x')
- virtual int **set_radius** (const int value, const int num=1)
- virtual int **set_caption** (const string content)
- virtual bool **highlight** (double amount=1.0)
- virtual bool **unhighlight** ()

Protected Types

- enum { **OBJ**, **TDS** }

Protected Member Functions

- int **loadobj** ()
- int **load3ds** ()
- int **parsemtl** (const string &mtlfilename, string &texdirname)

Protected Attributes

- enum objectloader::LoadableObject:: { ... } **filetype**
- string **filename**

Private Member Functions

- void **highlight_intern** (double amount=1.0)

Private Attributes

- bool **materials_doubled**
- shared_ptr< Material > **originalGlobalMat**
- shared_ptr< vector< Material > > **originalMaterials**

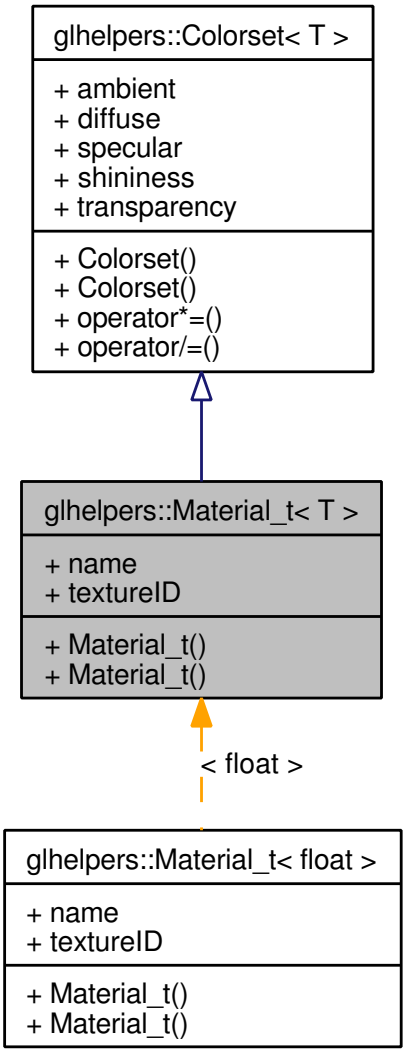
The documentation for this class was generated from the following files:

- objectloader.h
- objectloader.cpp

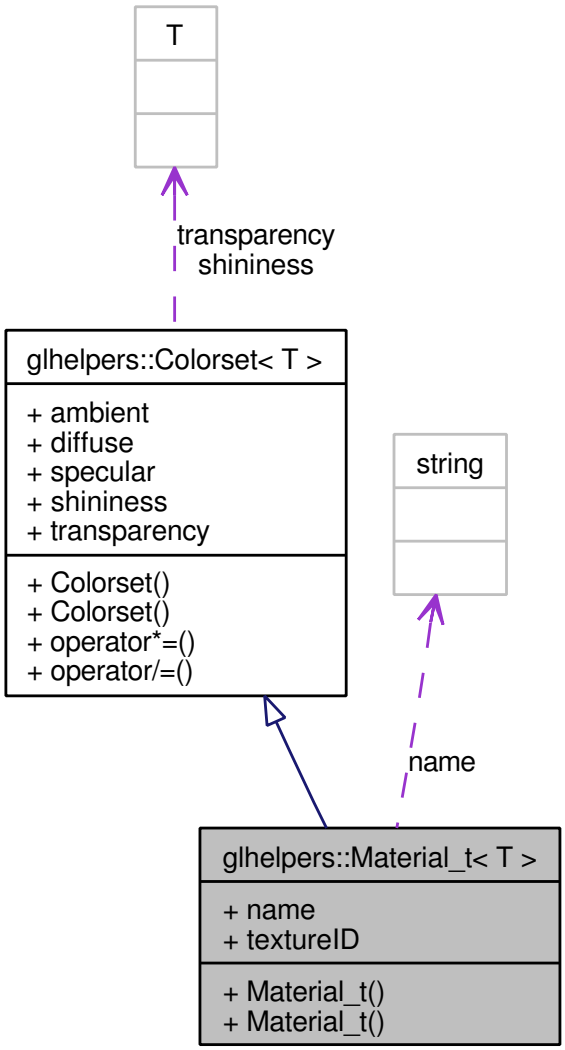
glhelpers::Material_t< T > Class Template Reference

`#include <glhelpers.h>`

Inheritance diagram for glhelpers::Material_t< T >:



Collaboration diagram for glhelpers::Material_t< T >:



Detailed Description

template<class T = float> class glhelpers::Material_t< T >

An OpenGL material containing a Colorset, a texture ID and a name.

Public Member Functions

- **Material_t** (const string &mtlname="")
- **Material_t** (const Material_t< T > &mat)

Public Attributes

- string **name**
- int **textureID**

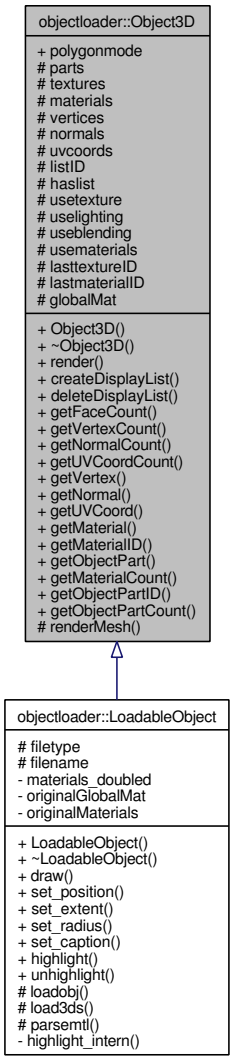
The documentation for this class was generated from the following file:

- glhelpers.h

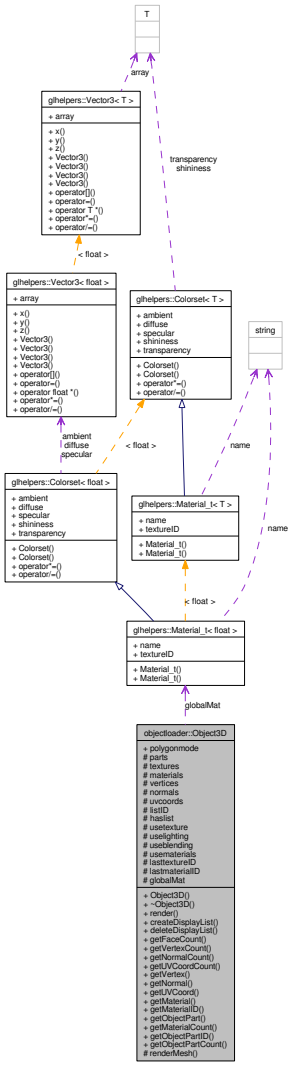
objectloader::Object3D Class Reference

`#include <objectloader.h>`

Inheritance diagram for objectloader::Object3D:



Collaboration diagram for objectloader::Object3D:



Detailed Description

An OpenGL mesh object.

It consists of one or more ObjectParts.

Public Member Functions

- void **render** (int partID=-1)
- void **createDisplayList** (int partID=-1)
- void **deleteDisplayList** (int partID=-1)
- int **getFaceCount** () const
- int **getVertexCount** () const
- int **getNormalCount** () const
- int **getUVCoordCount** () const
- Vector3f **getVertex** (int i) const
- Vector3f **getNormal** (int i) const
- const Vector2f **getUVCoord** (int i) const
- const Material & **getMaterial** (int id) const
- int **getMaterialID** (string name) const
- const ObjectPart & **getObjectPart** (int id) const
- int **getMaterialCount** () const
- int **getObjectPartID** (string name) const
- int **getObjectPartCount** () const

Public Attributes

- GLuint **polygonmode**

Protected Member Functions

- void **renderMesh** ()

Protected Attributes

- vector< ObjectPart > **parts**
- vector< Texture > **textures**
- shared_ptr< vector< Material > > **materials**
- vector< Vector3f > **vertices**
- vector< Vector3f > **normals**
- vector< Vector2f > **uvcoords**
- GLuint **listID**
- bool **haslist**
- bool **usetexture**

- bool **uselighting**
- bool **useblending**
- bool **usematerials**
- int **lasttextureID**
- int **lastmaterialID**
- Material **globalMat**

Friends

- class **ObjectPart**

The documentation for this class was generated from the following files:

- objectloader.h
- objectloader.cpp

ObjectList Class Reference

`#include <globallists.h>`

Detailed Description

This class stores a list of all graphObjects that are displayed.

The objecs are stored and referenced by their uniqID. If a given object is a 2D or ART object, you additionally have to store it in the FlatObjectList or the ARTObjectList respectively. You can get an instance of this class in any thread to get access to the global object list.

Public Types

- typedef icl::SmartPtr< icl::ICLDrawWidget3D::GLCallback, icl::PointerDelOp > **ObjectPtr**

Public Member Functions

- void clear ()
  Clears the list.
- void clear_publisher (const string &publisher_name)
  Deletes all objects created by given publisher.
- bool does_exist (const uniqID &id) const
  Checks whether the given uniqID already exists in the list.
- bool add (const uniqID &id, ObjectPtr &object)
  Adds an object to the list.
- bool remove (const uniqID &id)
  Removes the object with the given uniqID.
- graphObject * get_object (const uniqID &id)
  *Gets an ordinary (*not *smart) pointer to an object in the list.*

- void release_object () const

  This unlocks the mutex that is locked by get_object(const uniqID&).

- ObjectPtr get_smart_pointer (graphObject *obj)

  Gets a smart pointer to an object in the list you only have the raw pointer to.

- void draw_all_3D (ICLDrawWidget3D *widget) const

  This method calls graphObject::draw() for all 3D objects in the list.

Private Types

- typedef std::map< uniqID, ObjectPtr > **map_t**
- typedef std::map< uniqID, ObjectPtr >::iterator **iter_t**

Private Member Functions

- void **lock** () const
- void **unlock** () const

Static Private Attributes

- static map_t **object_list**
- static icl::Mutex **object_list_mutex**

Member Function Documentation

void ObjectList::clear () [inline]

Clears the list.

This removes all objects from the list, leaving it blank.

void ObjectList::clear_publisher (const string & *publisher_name*) `[inline]`

Deletes all objects created by given publisher.

Parameters:

publisher_name The name of the publisher whose object are to be removed from the list.

bool ObjectList::does_exist (const uniqID & *id*) const `[inline]`

Checks whether the given uniqID already exists in the list.

Parameters:

id The uniqID to check for.

Returns:

`true` if an object with this ID exists, `false` otherwise.

bool ObjectList::add (const uniqID & *id*, ObjectPtr & *object*) `[inline]`

Adds an object to the list.

Parameters:

id The uniqID of the object to add.

object A smart pointer to the graphObject to add.

Returns:

`true` if the object could be added successfully, `false` if an object with this uniqID already existed.

bool ObjectList::remove (const uniqID & *id*) `[inline]`

Removes the object with the given uniqID.

Parameters:

id uniqID of the object to remove.

Returns:

`true` if the object could be removed successfully, `false` if no such object exists in the list.

graphObject * ObjectList::get_object (const uniqID & *id*) `[inline]`

Gets an ordinary (*not* smart) pointer to an object in the list.

You **must always** call release_object when you are done with the object you grabbed with this method and **before** calling any ObjectList method again! Don't use this method unless you have to.

Parameters:

id The uniqID of the object to grab.

Returns:

A pointer to the object with the given uniqID.

See also:

release_object

void ObjectList::release_object () const `[inline]`

This unlocks the mutex that is locked by get_object(const uniqID&).

You must always call this method immediately when you're done using the pointer get_object(const uniqID&) returned.

ObjectPtr ObjectList::get_smart_pointer (graphObject * *obj*) `[inline]`

Gets a smart pointer to an object in the list you only have the raw pointer to.

Parameters:

obj A pointer to a graphObject.

Returns:

A smart pointer to the given graphObject.

void ObjectList::draw_all_3D (ICLDrawWidget3D * *widget*) const `[inline]`

This method calls graphObject::draw() for all 3D objects in the list.

Parameters:

widget The ICLDrawWidget3D to draw the objects in.

Here is the call graph for this function:

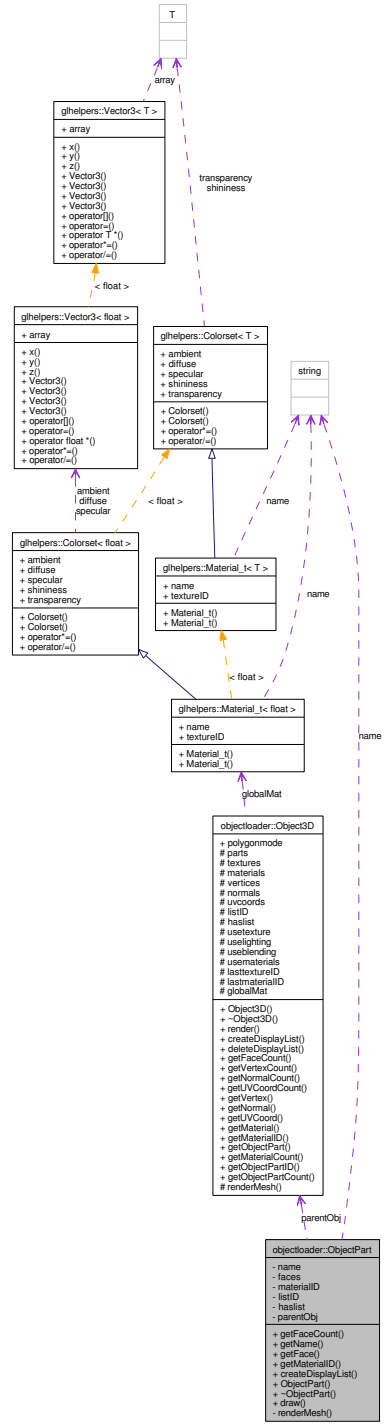
The documentation for this class was generated from the following files:

- globallists.h
- globallists.cpp

objectloader::ObjectPart Class Reference

`#include <objectloader.h>`

Collaboration diagram for objectloader::ObjectPart:



Detailed Description

One or more of these objects constitute one Object3D.

Public Member Functions

- int **getFaceCount** () const
- string **getName** () const
- const Face & **getFace** (int id) const
- int **getMaterialID** () const
- void **createDisplayList** ()
- **ObjectPart** (Object3D *parent)
- void **draw** ()

Private Member Functions

- void renderMesh ()

Private Attributes

- string **name**
- vector< Face > **faces**
- int **materialID**
- GLuint **listID**
- bool **haslist**
- Object3D * **parentObj**

Friends

- class **LoadableObject**
- class **Object3D**

Member Function Documentation

void ObjectPart::renderMesh () `[private]`

set material

render mesh

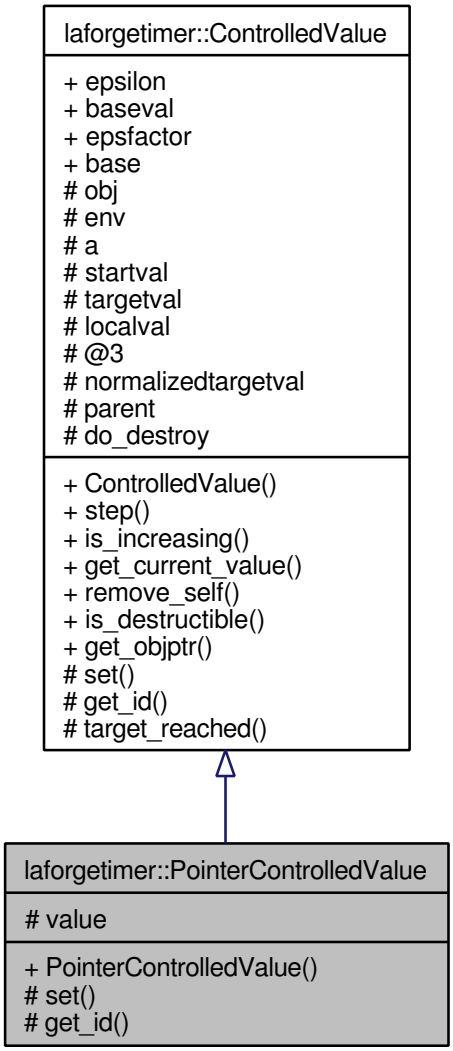
The documentation for this class was generated from the following files:

- objectloader.h
- objectloader.cpp

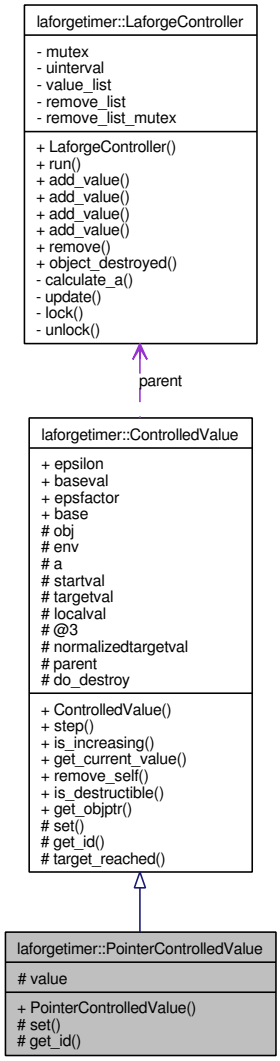
laforgetimer::PointerControlledValue Class Reference

`#include <laforgetimer.h>`

Inheritance diagram for laforgetimer::PointerControlledValue:



Collaboration diagram for laforgetimer::PointerControlledValue:



Detailed Description

This is a ControlledValue that controls a value directly via a pointer to a given variable.

Do not instantiate this class directly!

Public Member Functions

- **PointerControlledValue** (LaforgeController *parent_lc, ObjectList::ObjectPtr objptr, double *val, double target, double stepsize, envelope_t envelope, bool destructible)

Protected Member Functions

- virtual void **set** (double val) const
- virtual ValueID **get_id** () const

Protected Attributes

- double * **value**

The documentation for this class was generated from the following file:

- laforgetimer.h

ScottyCommunicator Class Reference

`#include <ScottySubscriber.h>`

Detailed Description

ScottySubscriber stores the values it gets from SCOTTY here.

They are stored in `static` variables so you can instantiate this class anywhere to access them. No mutexes or similar protection technique is used but the worst thing that should be able to happen is that you get half updated values which should result in something like an intermediate value for anything but very fast movements. If it turns out that movements get too edgy, a mutex should be used.

Author:

cmertes

Public Member Functions

- double getCenterX ()

  Gets the x component of the spot the camera currently looks at.

- double getCenterY ()

  Gets the y component of the spot the camera currently looks at.

- double getCenterZ ()

  Gets the z component of the spot the camera currently looks at.

- double getEyeX ()

  Gets the x component of the spot the camera is positioned at.

- double getEyeY ()

  Gets the y component of the spot the camera is positioned at.

- double getEyeZ ()

  Gets the z component of the spot the camera is positioned at.

– double getUpX ()

Gets the x component of the vector that points upwards in the camera coordinate system.

– double getUpY ()

Gets the y component of the vector that points upwards in the camera coordinate system.

– double getUpZ ()

Gets the z component of the vector that points upwards in the camera coordinate system.

– double getDirectionX ()

Gets the x component of the vector that's parallel to the direction the camera currently looks into.

– double getDirectionY ()

Gets the y component of the vector that's parallel to the direction the camera currently looks into.

– double getDirectionZ ()

Gets the z component of the vector that's parallel to the direction the camera currently looks into.

– void setEyeX (double d)

Sets the x component of the spot the camera is positioned at.

– void setEyeY (double d)

Sets the y component of the spot the camera is positioned at.

– void setEyeZ (double d)

Sets the z component of the spot the camera is positioned at.

– void setUpX (double d)

Sets the x component of the vector that points upwards in the camera coordinate system.

– void setUpY (double d)

Sets the y component of the vector that points upwards in the camera coordinate system.

– void setUpZ (double d)

Sets the z component of the vector that points upwards in the camera coordinate system.

- void setDirectionX (double d)

  Sets the x component of the vector that's parallel to the direction the camera currently looks into.

- void setDirectionY (double d)

  Sets the y component of the vector that's parallel to the direction the camera currently looks into.

- void setDirectionZ (double d)

  Sets the z component of the vector that's parallel to the direction the camera currently looks into.

Static Private Attributes

- static double **eyex** = 0.0
- static double **eyey** = 0.3
- static double **eyez** = 0.5
- static double **upx** = 0.0
- static double **upy** = 0.0
- static double **upz** = 0.0
- static double **dirx** = 0.0
- static double **diry** = 0.0
- static double **dirz** = -1.0

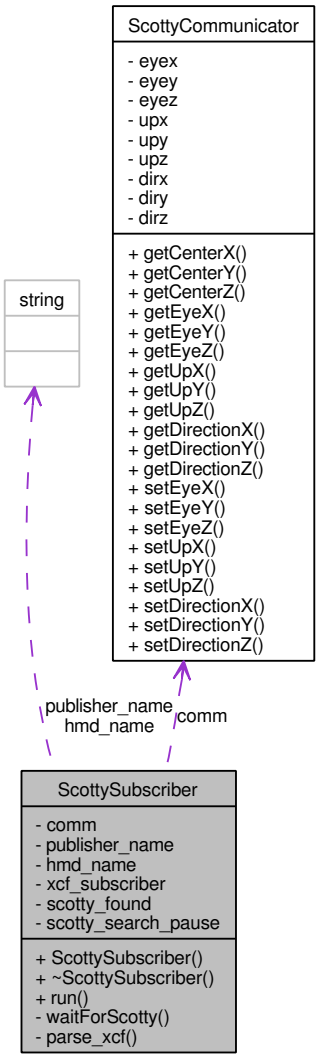
The documentation for this class was generated from the following files:

- ScottySubscriber.h
- ScottySubscriber.cpp

ScottySubscriber Class Reference

`#include <ScottySubscriber.h>`

Collaboration diagram for ScottySubscriber:



Detailed Description

Subscribes to SCOTTY to get the newest camera coordinate system.

An instance of this class is a new thread and must be started by start().

Thereafter the data in ScottyCommunicator are updated with every package from SCOTTY. If no running SCOTTY is found or an old one died, the Threads polls regularly for a new SCOTTY publisher (ScottySubscriber::waitForScotty()). Before the first data from SCOTTY arrived there is some default coordinate system; when a SCOTTY dies, the coordinate system is frozen to its last state.

Author:

cmertes

Public Member Functions

- ScottySubscriber (string publisherName, string hmdName)

  The class constructor.

- virtual void run ()

  Starts the thread.

Private Member Functions

- void waitForScotty ()

  If no SCOTTY is found or the connection to an existing SCOTTY died, this method is called to wait for a new SCOTTY to appear.

- bool parse_xcf (string message)

  When SCOTTY sent a new message, the data it contains is extraced by this function.

Private Attributes

- ScottyCommunicator **comm**
- const string **publisher_name**
- const string **hmd_name**
- XCF::SubscriberPtr **xcf_subscriber**
- bool scotty_found

this is `truewhen we are connected to SCOTTY`

Static Private Attributes

– static const unsigned int scotty_search_pause = 1000

time between trials to find SCOTTY in msecs

Constructor & Destructor Documentation

ScottySubscriber::ScottySubscriber (string *publisherName*, string *hmdName*)

The class constructor.

It takes the publisher name and the content of the name attribute of the hmd tag this thread should evaluate. Normally there are two HMDs, each with its own block of data. This constructor must be followed by start() for the thread to start.

Parameters:

publisherName The name of the SCOTTY publisher this thread should subscribe to.

hmdName The name of the HMD to pay attention to. Normally "hmd0" or "hmd1".

Member Function Documentation

void ScottySubscriber::waitForScotty () `[private]`

If no SCOTTY is found or the connection to an existing SCOTTY died, this method is called to wait for a new SCOTTY to appear.

It polls in intervals specified by ScottySubscriber::scotty_search_pause.

Here is the caller graph for this function:

bool ScottySubscriber::parse_xcf (string *message*) [private]

When SCOTTY sent a new message, the data it contains is extraced by this function.

It also directly sets the ScottyCommunicator accordingly.

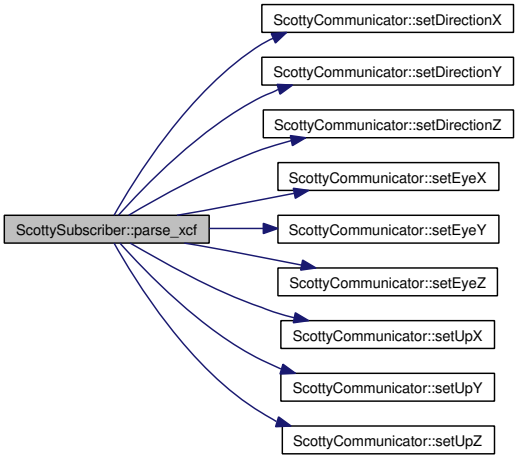
Parameters:

The string containing the message as XML

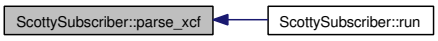
Returns:

`true` if everything went fine, `false` if something went wrong

Here is the call graph for this function:



Here is the caller graph for this function:



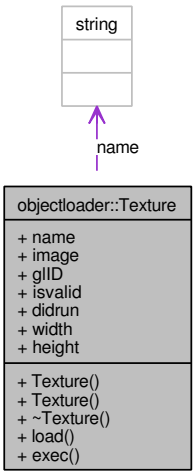
The documentation for this class was generated from the following files:

- ScottySubscriber.h
- ScottySubscriber.cpp

objectloader::Texture Class Reference

`#include <objectloader.h>`

Collaboration diagram for objectloader::Texture:



Detailed Description

A class representing a texture as referenced by its ID in a Material.

The image data is stored here, too.

Public Member Functions

- **Texture** (const string &path)
- int **load** (const string &filename)
- int **exec** ()

Public Attributes

- string **name**
- boost::shared_array< GLbyte > **image**
- GLuint **glID**
- bool **isvalid**

- bool **didrun**
- int **width**
- int **height**

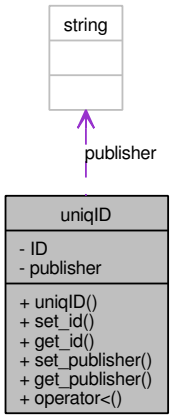
The documentation for this class was generated from the following files:

- objectloader.h
- objectloader.cpp

uniqID Class Reference

`#include <uniqid.h>`

Collaboration diagram for uniqID:



Detailed Description

An ID uniquely referencing graphical objects.

Each taibak generator, that is each publisher sending commands to LAFORGE, uses internally unique integers to reference the objects it created. As more than one publisher is allowed though and they shouldn't be forced to create globally unique IDs somehow, this class uses both integer IDs and the publisher name to identify objects. It uses fast integer comparison as long as this is unambiguous and falls back to slower string comparison when necessary. Taibak generators should therefore generate IDs randomly to make collisions rare if they care about performance but the class is designed to handle any combination of ID-publisher pairs as efficient as possible while keeping a strict total order on the set of uniqIDs.

To implement a set this class needs to be used in a container class though. It was designed with std::map in mind for this purpose.

Public Member Functions

- uniqID (const unsigned int id, const std::string &publ)

The class constructor takes the integer ID and the publisher name as arguments.

- void set_id (const unsigned int id)
  Resets the ID after creation.
- unsigned int get_id () const
  Gets the ID part of the uniqID.
- void set_publisher (const std::string &publ)
  Resets the publisher name after creation.
- std::string get_publisher () const
  Gets the publisher name part of the uniqID.
- bool operator< (const uniqID &u) const
  The comparison operator.

Private Attributes

- unsigned int **ID**
- std::string **publisher**

Member Function Documentation

bool uniqID::operator< (const uniqID & *u*) const [inline]

The comparison operator.

The ID part is given preference over the publisher name part.

Other comparison operators are not implemented but can be constructed from the lesser than operator if necessary.

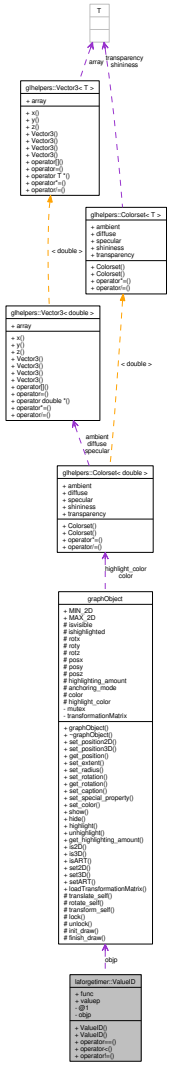
The documentation for this class was generated from the following file:

- uniqid.h

laforgetimer::ValueID Class Reference

`#include <laforgetimer.h>`

Collaboration diagram for laforgetimer::ValueID:



Detailed Description

This class uniquely identifies a ControlledValue.

This happens either by its value pointer or by its function pointer *and* its graphObject pointer.

Public Member Functions

- **ValueID** (double *valp)
- **ValueID** (value_function_t f, graphObject *obj)
- bool **operator==** (const ValueID &rhs) const
- bool **operator<** (const ValueID &rhs) const
- bool **operator!=** (const ValueID &rhs) const

Private Attributes

- union {
  value_function_t **func**
  double * **valuep**
  };

- graphObject * **objp**

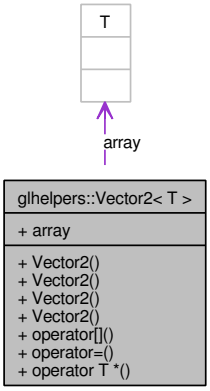
The documentation for this class was generated from the following file:

- laforgetimer.h

glhelpers::Vector2< T > Class Template Reference

`#include <glhelpers.h>`

Collaboration diagram for glhelpers::Vector2< T >:



Detailed Description

template<class T> class glhelpers::Vector2< T >

A simple two-dimensional vector.

Public Member Functions

- **Vector2** (T f1=0.0, T f2=0.0)
- **Vector2** (T *const a)
- **Vector2** (const Vector2< T > &a)
- **Vector2** (const vector< T > &v)
- T & **operator[]** (int i)
- Vector2 & **operator=** (const Vector2 &vec)
- **operator T** * ()

Public Attributes

- T **array** [2]

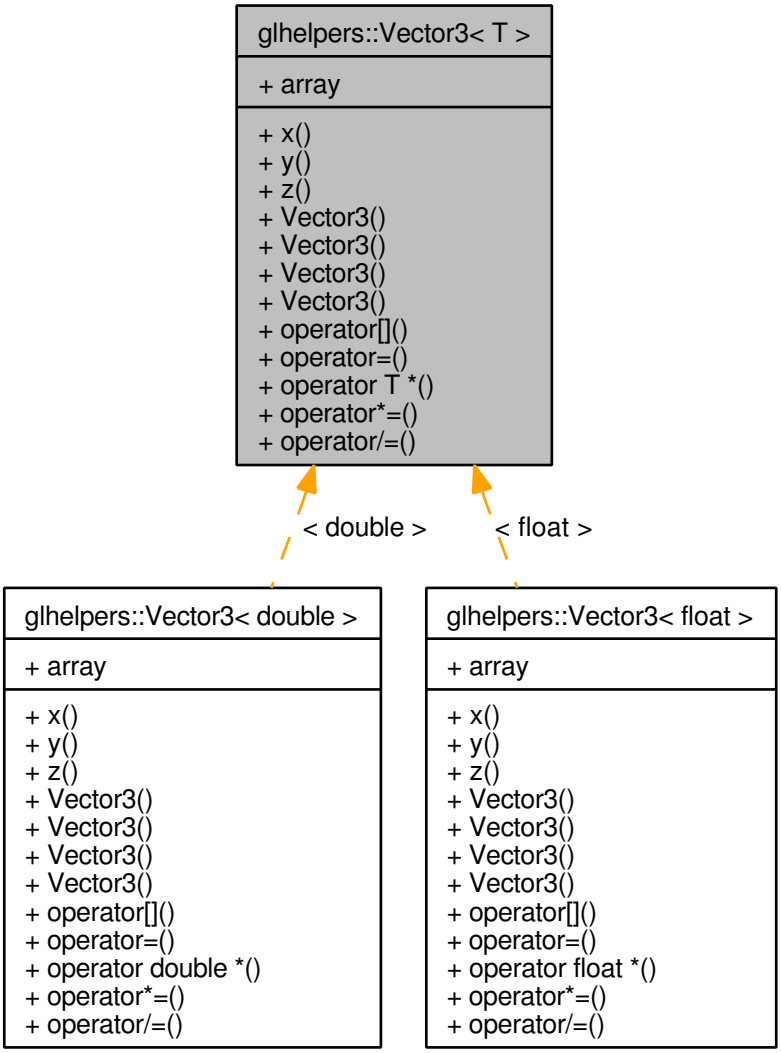
The documentation for this class was generated from the following file:

- glhelpers.h

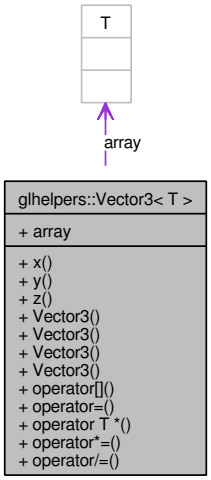
glhelpers::Vector3< T > Class Template Reference

`#include <glhelpers.h>`

Inheritance diagram for glhelpers::Vector3< T >:



Collaboration diagram for glhelpers::Vector3< T >:



Detailed Description

template<class T> class glhelpers::Vector3< T >

A simple three-dimensional vector.

Public Member Functions

- T & **x** ()
- T & **y** ()
- T & **z** ()
- **Vector3** (T f1=0.0, T f2=0.0, T f3=0.0)
- **Vector3** (T *const a)
- **Vector3** (const Vector3< T > &a)
- **Vector3** (const vector< T > &v)
- T & **operator[]** (int i)
- Vector3 & **operator=** (const Vector3 &vec)
- **operator T** * ()
- Vector3< T > & **operator*=** (const Vector3< T > &rhs)
- Vector3< T > & **operator/=** (T rhs)

Public Attributes

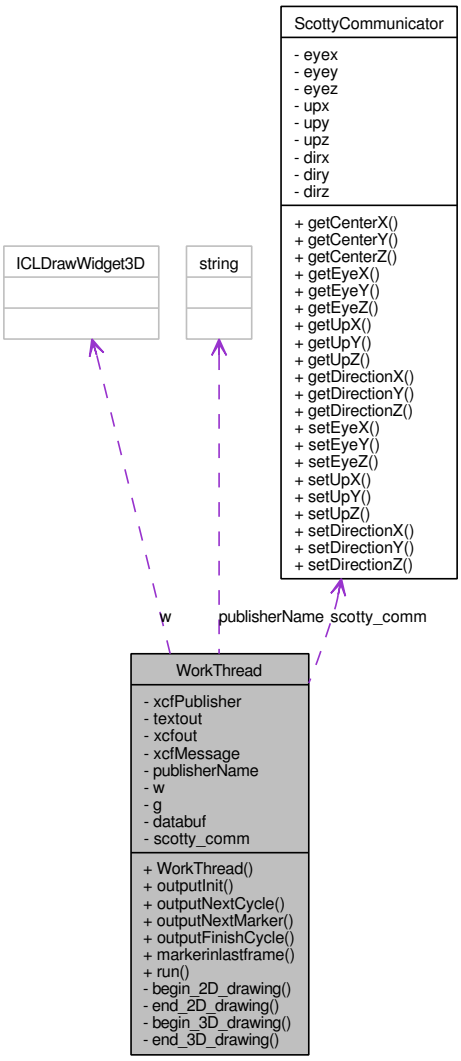
- T **array** [3]

The documentation for this class was generated from the following file:

- glhelpers.h

WorkThread Class Reference

Collaboration diagram for WorkThread:



Detailed Description

Contains the program's main visualization loop.

Author:

celbrech

Public Member Functions

- void **outputInit** ()
- void **outputNextCycle** ()
- void **outputNextMarker** (int id, float posX, float posY, float posZ)
- void **outputFinishCycle** ()
- bool **markerinlastframe** (iclart::ARMarkerInfo *mi_last, int id, int n_last)
- virtual void **run** ()

Static Private Member Functions

- static void **begin_2D_drawing** (void *)
- static void **end_2D_drawing** (void *)
- static void begin_3D_drawing (void *)

  Initializes the projection and modelview matrices to set up the virtual camera for 3D objects (not ART or 2D objects).

- static void **end_3D_drawing** (void *)

Private Attributes

- XCF::PublisherPtr **xcfPublisher**
- bool **textout**
- bool **xcfout**
- std::ostringstream **xcfMessage**
- string **publisherName**
- ICLDrawWidget3D * **w**
- Grabber * **g**
- icl8u * **databuf**

Static Private Attributes

- static ScottyCommunicator **scotty_comm** = ScottyCommunicator()

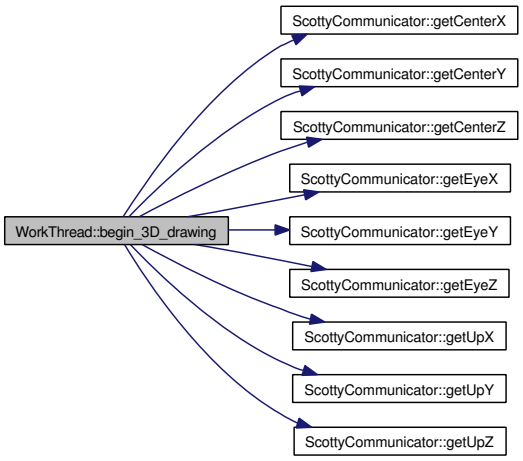
Member Function Documentation

static void WorkThread::begin_3D_drawing (void *) `[inline, static, private]`

Initializes the projection and modelview matrices to set up the virtual camera for 3D objects (not ART or 2D objects).

The projection matrix is set according to some camera dependent constants and the formula $\alpha_y = 2 \arctan(\frac{d_y}{2f})$ with α_y being the vertical FOV, d_y the vertical sensor size (e.g. its vertical resolution) and f the focal length in the same unit as d_y (usually pixels). The modelview matrix is initialized with data from ScottyCommunicator. This function is meant to be passed to ICLDrawWidget3D::callback(GLCallbackFunc, void *data) which explains the unused `void *` parameter.

Here is the call graph for this function:



The documentation for this class was generated from the following file:

- laforge.cpp

XCFThread Class Reference

`#include <intercomm.h>`

Collaboration diagram for XCFThread:

Detailed Description

This thread is created by LaforgeInterceptor to listen to a single XCF publisher sending {Taibaks} (packets containing control commands, for example by PICARD).

When it receive such a packet it parses its content and modifies the LAFORGE object lists and objects accordingly.

Public Member Functions

- XCFThread (const string &name, LaforgeInterceptor &parent)

  The class constructor.

- ~XCFThread ()

  Due to some internal problems of XCF this destructor has to sleep a certain amount of time.

- virtual void run ()

  This method is run when the thread is started.

- string getName ()

  Gets the publisher name this thread listens to.

Private Types

- enum **cmd_t** {
  CMD_CREATE, **CMD_MODIFY**, **CMD_DELETE**, **CMD_CLEAR**,
  CMD_LOGGING }

Private Member Functions

- bool **parse_xcf** (string &taibak)
- bool **parse_create** (ObjectList::ObjectPtr &obj_p, const Location &object_location, const uniqID &curr_uniqid)
- bool **parse_envelope** (const Location &env_parent_location, laforgetimer::envelope_t &env, double &a, double &t, bool &use_-t)

- bool **parse_object_children** (graphObject *obj, ObjectList::ObjectPtr &obj_p, const Location &object_location)
- bool **parse_effects** (ObjectList::ObjectPtr objptr, graphObject *obj, const Location &effects_location)
- shared_ptr< vector< double > > **parse_matrix** (const Location &matrix_location, const size_t numrows, const size_t numcolumns)
- void **clear_content** ()

Private Attributes

- const string **publisher_name**
- LaforgeInterceptor & **parent_interceptor**
- XCF::SubscriberPtr **xcf_subscriber**
- ARTObjectList **art_object_list**
- list< graphObject::artID > **artidlist**
- ObjectList **object_list**
- FlatObjectList **flat_object_list**

Constructor & Destructor Documentation

XCFThread::XCFThread (const string & *name*, LaforgeInterceptor & *parent*) `[inline]`

The class constructor.

Parameters:

name The name of the publisher to listen to.

parent The LaforgeInterceptor that created this thread.

XCFThread::~XCFThread () `[inline]`

Due to some internal problems of XCF this destructor has to sleep a certain amount of time.

This is highly error prone and should be changed as soon as XCF provides an alternative solution!

Member Function Documentation

string XCFThread::getName () `[inline]`

Gets the publisher name this thread listens to.

Returns:

The name of the XCF publisher assigned to the current XCFThread instance.

The documentation for this class was generated from the following files:

- intercomm.h
- intercomm.cpp
- intercommold.cpp

D Questionnaire

Fragebogen zur Anzeige des Aufmerksamkeitsfokus

Geschlecht _____ Alter _____

Bitte beschreibe kurz in eigenen Worten deinen ersten Eindruck beim Benutzen des Systems.

Hattest du vor diesem Versuch schon einmal ein Augmented-Reality-System benutzt?	Ja			Nein	
	O			O	
	Gar nicht störend				Sehr unangenehm
Wie unangenehm fandest du das System in diesem Versuch?	O	O	O	O	
Falls du es unangenehm fandest: Wie hat sich der fehlende Komfort geäußert?					
	Sehr schnell				Gar nicht
Wie schnell hast du dich an das System gewöhnt?	O	O	O	O	
	Sehr stark				Gar nicht
Wie stark hast du die visuelle Hervorhebung genutzt?	O	O	O	O	
	Sehr stark				Gar nicht
Wie stark hast du die auditive Hervorhebung genutzt?	O	O	O	O	
	Sehr hilfreich			Gar nicht hilfreich	Störend
Wie hilfreich fandest du die visuellen Hervorhebungen?	O	O	O	O	O
	Sehr hilfreich			Gar nicht hilfreich	Störend
Wie hilfreich fandest du die auditiven Hervorhebungen?	O	O	O	O	O

Was hat dir besonders gut, was besonders schlecht gefallen? Was würdest du ändern?

Unter welchen Bedingungen – wenn überhaupt – könntest du dir vorstellen, ein solches System im Alltag tatsächlich einzusetzen?

Vielen Dank für deine Hilfe!

Glossary

alignment: A linguistic term describing the implicit mutual convergence of the internal states of dialogue partners through very basic mechanisms. This can happen on different levels, between different subsystems. The logical connection between these subsystems is called an alignment channel.

alignment channel: Alignment needs similar information-processing systems on both sides of a communication which can each consist of several subsystems. These subsystems can align more or less independently, even when hierarchically organized. Therefore one can imagine direct connections on an abstract level between subsystems that do not talk directly to each other. These are called alignment channels and there can be as many as there are subsystems. In principle not more than one modality with one communication channel is needed to bear all existing alignment channels, although that depends on the communication channel and the modality used.

AR: Augmented reality. The addition of computer-generated virtual entities to the real world. This can happen through many means, from semi-transparent displays or similar hardware up to projecting them onto real-world objects.

ART: Augmented Reality Toolkit, also AR Toolkit. A system that uses the distortion and size of square markers to calculate their orientation in space and identifies them by a pattern inside the square. Like that 3D models can easily be made to look as if they were sitting on top of the markers.

blackboard system: Describes a common "space" where information can be put by processes and then be picked up by others. The processes do not have to know of each other and they can ask the blackboard system to retrieve the information they need.

C5: A project within the Collaborative Research Centre *Alignment in Communication* at Bielefeld University called *Alignment in AR-based cooperation*. It aims to provide means for the investigation of alignment by using augmented reality.

communication channel: A way by which interlocutors encode information. The most obvious and arguably most important communication channel is the verbal one. Gesture, facial expressions and prosody also play important roles.

DOF: Degree of Freedom. Every translation along or rotation around an axis is an additional degree of freedom if it is independent of the others. So coordinates for the axes x, y and z constitute three degrees of freedom. Adding the position in terms of $-x$ does not increase this number, nor does any other linear combination of the first three axes. Rotations around these axes, however, do add additional information and hence they add to the number of degrees of freedom.

FOV: Field of view. Designates either the whole volume seen by a person or a camera, or the area on a plane of interest (a table in our setup). It can also mean the angle between the planes that restrict the volume seen (which is then sometimes called the *frustum*). For cameras with their rectangular sensors there is a horizontal and a vertical field of view in this sense of the word.

GLUT: OpenGL Utility Toolkit. Arguably the most important, cross-platform extension library for OpenGL.

HMD: Head-mounted display. A computer display that can be attached to someone's head so he does not have to move his head to look at it. There are transparent and opaque forms of this.

LAFORGE: The software module that visualizes the camera images and augments them according to commands it received via XCF.

medium: A way by which encoded information is transmitted or stored. In prehistoric times there was but the air between people, later the written word was invented and in recent times a great number of electric and electronic media followed.

modality: Sensory modalities in the context of this work are the input channels people have at their disposal. Although sight is said to be the most important modality for humans, for communication hearing is probably much more important.

OpenGL: Open Graphics Library. A programming interface for 3D graphics. It is supported natively by almost all modern graphics hardware.

OSC: Open Sound Control. A protocol to transmit data in a fast, binary way.

PICARD: The software module that controls LAFORGE and provides the interface to the experimenter.

publish/subscribe: This is a paradigm for messaging between processes. Any process can provide information and *publish* it while other processes can *subscribe* to it. The processes themselves do not have to deal with whom they are sending the data to or where they are fetching it from.

VR: Virtual reality. The replacement of the normal world around us with a purely computer-generated one in the perception of a user.

XCF: XML enabled Communication Framework. A XML-based publish/subscribe system.

XML: Extensible Markup Language. XML is a standard for creating custom markup languages. This means that text can be extended by so called *tags* which stand on their own or enclose some other text or further text. They can also have *attributes* to provide further details about them. So the natural representation of XML documents is a tree.

Acknowledgements

I want to express my gratitude to many people whose support I can hardly value adequately through these lines. First and foremost this applies to the C5 team member and my dear friend Angelika Dierker who provided me with her expertise, her counsel, her friendship and too much of her time for me to feel comfortable contemplating it.

The second woman I want to thank is actually the one that comes first in my life and I dare not think about how much more desperate the difficult times I had during the writing of this work would have been without the constant loving support of Daniela Leichsenring. Her patience with me was admirable and surpassed my own by far.

I would further like to thank Marc Hanheide and Thomas Hermann who slipped me into their tight schedule more often than it allowed, and whose advice and knowledge proved invaluable every time they did. It is also they who made the C5 project what it is, so – although it is a much used phrase – this work would not exist if it were not for them.

Christof Elbrechter and Alexander Neumann unhesitatingly sacrificed time they should have spent on their work for mine, again and again, which I feel very thankful though guilty about. Christof's knowledge of the ICL and his debugging expertise are beyond price and Alex had a way of "listening me into thinking" that was almost as staggering as his tenaciousness when trying to get SCOTTY to work properly while he should have been writing his own Bachelor's thesis.

I am also very grateful to Marcel Martin, Jan Stallkamp and Steve Wolter who each proved a keen eye for details as well as for the greater picture when proofreading this work and by this improved its quality a lot through many valuable remarks. Stefan Vitz even offered me this service without me begging him for it and I regret not to have provided him with an op-

portunity to do so early enough for him to proofread this whole work as distinguishedly as he did with the first chapter.

Furthermore, I should like to thank Till Bovermann, René Tünnermann and Eckard Riedenklau for their constant availability for questions and their untiring willingness to answer them in the most helpful of ways.

And finally, thank you to each of the sixteen kind persons who agreed to be a subject in my study for a mere candy bar.

I should like to address a further note of acknowledgement to the Centre of Excellence 673 *Alignment in Communication* and thereby to the German Research Foundation (*Deutsche Forschungsgemeinschaft*) who kindly provided the technical and personal resources to conduct this research.

Declaration

Bielefeld, 31 August 2008

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

Christian Mertes