Augmented Visualization with Natural Feature Tracking

Gábor Sörös
Swiss Federal Institute of Technology Zurich
Universitaetstrasse 6
8092 Zurich, Switzerland
gabor.soros@inf.ethz.ch

Peter Rautek
Vienna University of Technology
Favoritenstrasse 9-11
1040 Vienna, Austria
rautek@cg.tuwien.ac.at

Hartmut Seichter
Graz University of Technology
Inffeldgasse 16
8010 Graz, Austria
seichter@icg.tugraz.at

Eduard Gröller
Vienna University of Technology
Favoritenstrasse 9-11
1040 Vienna, Austria
groeller@cg.tuwien.ac.at

ABSTRACT

Visualization systems often require large monitors or projection screens to display complex information. Even very sophisticated systems that exhibit complex user interfaces do usually not exploit advanced input and output devices. One of the reasons for that is the high cost of special hardware. This paper introduces *Augmented Visualization*, an interaction method for projection walls as well as monitors using affordable and widely available hardware such as mobile phones or tablets. The main technical challenge is the tracking of the users' devices without any special equipment or fiducial markers in the working area. We propose to track natural features of the display content with the built-in camera of mobile devices. Tracking the visualized scene allows pose estimation of the mobile devices with six degrees of freedom. The position and orientation information is then used for advanced interaction metaphors like *magic lenses*. For a group of experts who are analyzing the data in front of the same screen, a personal augmented view of the visualized scene is presented, for each user on his/her personal device. The prototype *Augmented Visualization System* achieves interactive frame rates and may lead to a greatly enhanced user experience. The paper discusses the design and implementation questions and illustrates potential application scenarios.

Categories and Subject Descriptors

I.3.6 [**Computing Methodologies**]: Computer Graphics—*Interaction Techniques*; I.4.9 [**Computing Methodologies**]: Image Processing and Computer Vision—*Applications*

General Terms

Design, Experimentation, Human Factors

MUM'11, Dec 7–9, 2011, Beijing, China.

Keywords

Handheld augmented reality, natural feature tracking, interactive visualization systems, human computer interaction

1. INTRODUCTION

In scientific visualization, usually a group of individuals or experts analyze a scene on a large monitor or projection screen. During the discussion the participants may have different interests regarding their field of specialization. For example, in the medical domain a surgeon, a cardiologist, and a neurologist may want to investigate different aspects of the underlying data. We propose the use of their personal mobile devices (e.g., smartphones or tablet PCs such as an iPad) to interact with the visualized scene and to explore user-specific information that is not visible on the common screen.

Our prototypic implementation extends an existing volume visualization system. It processes volumetric data sets and renders 2D views of the content. In addition to the common large-screen visualization, users can see their own augmented renderings on their mobile devices according to their interests. To get an enhanced view of the discussed data, the users have to orient their camera-equipped handheld devices towards the screen. The augmenting elements are spatially registered to and superimposed on the original common display content. For interaction, the touch screens of the mobile devices are used and the movements of the devices themselves can also be mapped to actions. The positions of the mobile devices are estimated relatively to the common screen. This is done by comparing local feature correspondences of the rendered and the captured images using state-of-the-art computer vision techniques. The novelty in our approach is to track the common display's dynamic content, i.e., the changing visualized scene itself. We targeted an affordable setup without any high-priced tracking devices, therefore, we focused on pure visual tracking using off-the-shelf mobile devices with built-in camera. We aimed at finding a markerless approach because passive markers are obstructive and lead to lower user acceptance, while active markers would add unnecessary complexity to the visualization setup. Instead, we chose a pure software-based solution that does not need any additional special hardware and does not change any aspect of existing visualization systems. The only assumption we made is that the discussed content is a planar image which can change regularly in time, but not at very short intervals. The core of our system is a

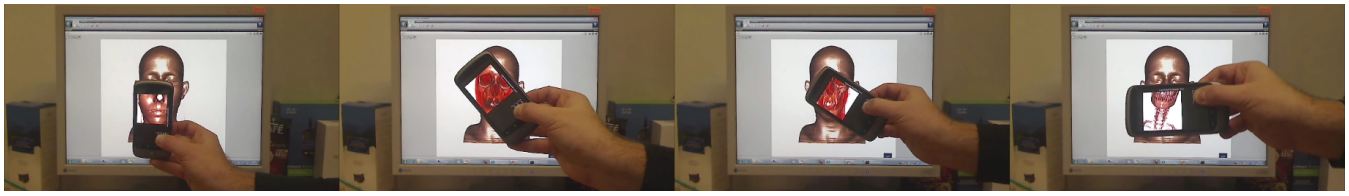


Figure 1: The magic lens metaphor in augmented visualization

tracker module that performs natural feature tracking on the video input of the user devices. Through the redundancy in natural feature tracking, we actually gain better robustness against lighting changes and occlusion. The system prototype presented in this paper requires close-to-real-time video streaming between the mobile client and the tracker. We also propose modifications of the system architecture that would allow us to replace video streaming with more compact data streaming towards the mobile clients. The realization of the second approach will be possible when the applied visual tracker component becomes available on the selected mobile platform. The estimated six-degrees-of-freedom pose information and other input events (e.g., button or touch-screen events) are transmitted back to the rendering system, which creates a personal augmented view of the scene for each user. The personal augmented image is presented on top of the camera preview at the user side. Fig. 1 and Fig. 2 show illustrations of the idea. We named the approach *Augmented Visualization*, where the term augmented refers to Augmented Reality (AR), a technique that overlays virtual objects and information on the real world to enhance human visual perception.

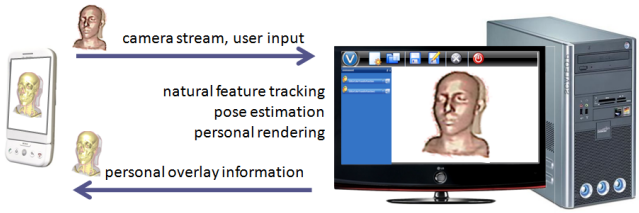


Figure 2: Illustration of the Augmented Visualization System

This paper focuses on the realization of an Augmented Visualization System. Section 2 gives a short overview of recent related projects, followed by several potential applications illustrated in Section 3. Section 4 deals with the design and implementation questions of our system introducing the main components in detail. Section 5 presents applications realized with the prototype together with performance measurements, while Section 6 discusses the limitations of the proposed technique and concludes the paper.

2. RELATED WORK

In recent years, several projects have dealt with interaction design for ubiquitous computing environments and some of them have applied visual localization approaches. Slay and Thomas [26] describe a universal interaction controller for in-situ visualization of ambient information across multiple heterogeneous displays. In their scenario, mobile devices are connected with public displays and take the role of a controller to interact with the content shown on the displays. A mobile device can also act as a clipboard, to temporarily store information, and transfer it between different displays. In an earlier work, Slay et al. [25] apply artificial markers to interact with a virtual scene in a very intuitive way, by showing special markers for the camera as commands. Jeon et al. [16] implemented different interaction scenarios for a group of people at a large display using camera-equipped mobile phones and marker-tagged objects on the screen. In both papers, authors use the ARToolKit library [2] for fiducial marker detection, which is a predecessor of the augmented reality framework applied in our system.

During the last years, the recognition of visual markers with mobile phones has become a widespread technology for interaction with real world objects. The information gathered through mobile image processing serves as a physical hyperlink to access actual object-related information [6]. Ballagas and Rohs [7] developed a mouse-click function for large screens using ghosted fiducial markers. They appear on a regular grid on the screen, at the time the user clicks with the mobile phone. The recorded image is used to localize the position of the click. A significant drawback of this method is the use of additional 2D barcodes to determine the position of the camera. Further, this method is only meant to use the mobile phone like a computer mouse in order to drag and drop elements on the screen. This is the objective of the DirectPointer [17] system as well but with the breakthrough of avoiding artificial markers.

In 2007, Boring et al. [10] presented the Shoot & Copy technique for recognizing icons on advertisement displays. The user simply takes a picture of the information of interest, and the system then retrieves the actual data represented on the screen, such as a stock quote, news text, or a piece of music. The technique does not require visual codes that interfere with the shown content. The captured image region is analyzed on a server (the display's host computer), which compares image patches to its own screen content and identifies the captured region. A reference to the corresponding data is sent back to the mobile phone. Once the user has time to view the information in more detail, the system allows to retrieve the actual data from this reference. This is the first approach to use pieces of screen content as markers. However, the recognition procedure is fairly slow and is limited to previously stored and analyzed advertisements.

Quack et al. [19] present a slide-tagging application for smart meeting rooms. The users have the possibility in employing their camera phones to click on slides or sections of slides, that are being presented on a projection screen. The user simply takes a photo of the current slide with the mobile phone, and the phone sends a query to the processing server, where scale-invariant local features are extracted from the photo. Then for each feature a nearest-neighbor search in the reference database of the presentation's slides is executed. The resulting potential matches are verified using projective geometry constraints. This way the actual slide of the presentation can unambiguously be determined. The user gets the information present on the slide to record it for his/her notes or to add tags.

The TouchProjector [11] is an approach to interact with a distant display through live video. It transforms a mobile phone's

touch-screen events to camera rays and enables interaction with intersected objects shown on the display. The drawback of the implementation lies in the pose estimation algorithm which is restricted to rectangle-shaped static objects (e.g., photos) on the screen.

Our interaction idea was inspired by recent works of Sanneblad and Holmquist [24] and Kalkofen et al. [18]. They used the *magic lens* metaphor (originally from Bier et al. [9]) in graphics and augmented reality applications to discover hidden structures.

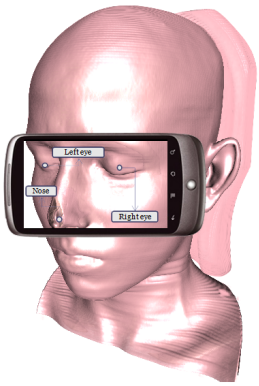
We neither want to tag the projected scene nor the displaying screen with any artificial markers, therefore we apply natural feature tracking (NFT), a markerless visual tracking solution. Its particular advantages are that no special hardware equipment and no changes are needed in the working area. Arbitrary-shaped natural objects are detected and collected to build a knowledge database of the screen content. The most successful NFT methods are based on sparse features such as scale-invariant interest points and local descriptors. The realization of our interaction method became possible by applying the natural feature tracking algorithm described by Wagner et al. [29] [30] within an augmented reality framework developed at the Christian Doppler Laboratory for Handheld AR.

3. APPLICATION SCENARIOS

We have defined different interaction scenarios from which two volume visualization applications have been realized until now. We combine the discussed hidden data exploration and controller perspectives in one technique, and apply the magic lens metaphor in which the handheld device corresponds to the lens. Fig. 1 shows an 'X-Ray Vision' application. In volume rendering, transfer functions define the color and opacity values of volume elements based on their density value. Thus, a transfer function determines what is shown and how it is colored on the rendered image. For the 'X-Ray' scenario the position and orientation estimation of the client device must be very accurate. The calculated pose matrix is combined with the modelview matrix of the original rendering, and subsequently with an additional transformation component (e.g., zooming). The generated overlay image depicts the same volume dataset but rendered using a different transfer function to see the bones inside the body. It appears to the user as if the mobile camera could see through the skin. If the viewpoint of the original model changes, the overlay has to change accordingly. It is also possible to modify specific rendering parameters depending on the position or orientation of the camera. The mobile device could be turned into a preview lens, for instance, by interpolating between two or more transfer functions while it is being rotated or while the user is walking from the left to the right of the screen. An orientation-sensitive menu as described by Adelmann [6] can easily be realized this way.

If the volume data is accompanied by segmentation information (for example, one knows which voxel belongs to which organ of the patient in a medical dataset) then direct scene annotation becomes possible. Rays are cast from the known user position into the volume and labels annotate the scene at intersections with objects (see Fig. 3(a)). In a similar use case there is no overlay at all, the user can see the camera preview on the mobile screen. If the screen is touched, a ray is cast into the volume and the user can interact with the data (select, brush, etc.) at a specific location, similarly to the work of Boring et al. [11].

The idea is not limited to volume rendering applications only. The screen content can originate from any visualization software starting from cartographic applications (Fig. 3(b)) through architectural visualizations to visual analytics. The common aspect in these applications is that they can contain hidden states (e.g., the layout of a pipe infrastructure, tourist attractions, etc.) that can be explored through the lens but take up no permanent screen space [9] [20]. Furthermore, the generated images are very rich in visual features, which is of particular importance for natural feature tracking.



(a) Direct scene annotations

(b) Cartographic Visualization

Figure 3: Application concepts (mock-ups)

Like the content, the physical setup is also relatively free to chose. Our system is applicable around a tabletop display without any modification. The PaperLens system [28] offers an alternative to the presented approach for tabletops. It has the advantages of supporting multiple users with lenses and higher resolutions while relying on passive IR-markers in a controlled environment. Our system, on the other hand, does not need any extra markers and is portable to practically any display. We envision multiplayer board games where the tabletop screen, which acts as a dynamic board, tells stories and plays animations during the game, while each player can see self-related hidden objects on the board by looking through his mobile phone.

The following section first gives an overview on the basic considerations of designing such an augmented visualization system, and then the three main components are described in detail.

4. DESIGN AND IMPLEMENTATION

Visualization, e.g., volume rendering, is a computationally demanding task. Therefore, the *rendering component* needs to run on a special machine, i.e., a high-end PC, to achieve interactive frame rates. If the overlay image is a different view of the same data, it is advantageous to render it on the same machine. We aim at

minimizing the dependencies between the rendering part and the other parts of the system to make the visualization software easily interchangeable.

The core task is the reliable tracking of the mobile devices, because their positions determine the overlays to be rendered and their motions are translated to interactions with the scene. We need to track them with six degrees of freedom, i.e., three translation parameters x, y, z and three orientation parameters *yaw*, *pitch*, *roll*, to know each client's viewpoint. The tracking system is required to capture the movement trajectory and deliver the current pose of each device close to real time. A built-in camera and a fast enough CPU are nowadays available on most mobile devices, making them suitable for computer vision approaches.

The quality of vision-based tracking highly depends on the camera and image sensor characteristics such as lens distortion, image resolution, or frame update rate, which tend to be rather poor on mobile devices. Inside-out visual tracking, where the camera is the object or is attached to the object being tracked, could be enhanced by fusing the vision-based tracking with other on-board sensor inputs such as a gyroscope, accelerometer and compass. We consider the implementation of a hybrid tracking approach to be out of the scope of this paper. Efficient and robust techniques for tracking fiducial markers do exist. However, the use of artificial black and white patches or retro-reflective markers is invasive and especially in the context of volume visualization unacceptable. With contemporary advances in technology it is possible to track an arbitrary previously known planar textures, if they contain enough distinctive features. In the computer vision literature this is referred to as natural feature tracking (NFT) or model-based planar tracking. Since this technology allows real-time tracking and 6DoF pose estimation without any intrusive changes to the visualization system, we chose to apply this technique.

To keep the system modular, we define the *tracker component* to be separate from the other parts. It deals with the necessary computer vision algorithms: feature detection, feature description, feature matching, outlier detection, pose estimation, and patch tracking. First, distinctive keypoints, e.g., corners, of the reference image are extracted. The patches around the keypoints are described as feature vectors and a database of features is built. Then, similarly, features of the captured image are extracted and described, and these feature vectors are compared to the database elements. If one finds at least three matches, the relative transformation (homography) between the two images can be estimated. Using the homography and the camera parameters the pose of the camera relative to the target can be calculated. As the target is the visualization screen, a local coordinate system between the screen and the mobile device can be established. The input to the tracker component are the reference images and the camera images. The output is a 4×4 pose matrix. The typical resolution of a projected reference image is about $1024x768$ pixels, and a typical captured camera image is about $320x240$ pixels. As we want to compare these two continuously changing image streams, i.e., the interactively rendered scene and the captured video, in a wireless infrastructure, it turns out to be beneficial to run the tracker on a PC and not on the mobile device. The video images are much smaller than the reference images and therefore data transfer is reduced. Another option to reduce bandwidth needs would be to extract the features of the captured image on the mobile device, and to transfer only the extracted features over the wireless network. Finally, we decided to use a PC-side tracker and stream the camera preview from the mobile devices to the PC. With this approach it is possible to exploit existing tracking libraries for PCs and to avoid a dependency on a specific mobile device model.

The *mobile component* incorporates all the user-interaction features and is treated as the third major part. For data transfer in such an indoor use case, IEEE 802.11b/g WiFi connection with up to 54Mbit/s data bandwidth is the most suitable one among the available features on a common smartphone. A drawback of our prototype approach is that because of the wireless video streaming, the number of clients is limited by the connection bandwidth.

Fig. 4 shows the main system components of the augmented visualization system and indicates their functions in temporal order. A scene is rendered and presented on the common screen and simultaneously sent to the tracker for analysis. The mobile device captures the screen content and continuously streams it to the tracker. The tracker component extracts features from the camera images and searches for matches within the database of the reference features. Then, it estimates the relative pose from feature correspondences. The pose information is sent back to the mobile device and to the rendering component to enhance user interaction and to render the personal overlay. The overlay imagery is presented on the user device. The three major building blocks are described in more detail in the following sections.

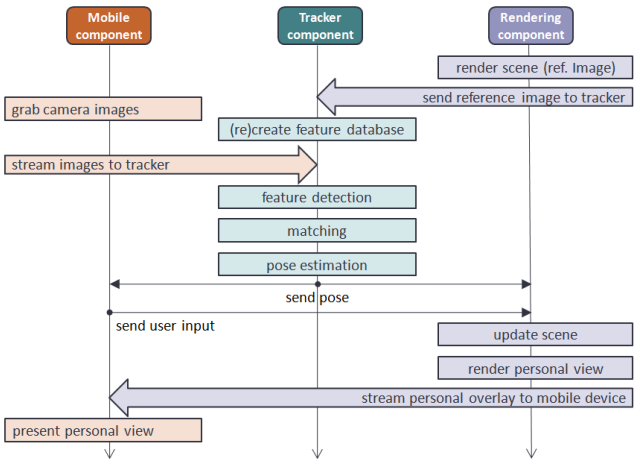


Figure 4: The three main components and their operation steps in our augmented visualization system

4.1 Rendering Component

Our prototype is implemented in C++. For testing and showcasing augmented visualization, we chose to integrate our software modules in a volume rendering framework named VolumeShop [12]. It contains several plugins for visualization-research purposes. All plugin parameters are stored in an XML file, which describes the whole visualization session. The parameters are potential targets of our investigation to be changed according to the movements of the mobile device. The properties of different plugins and framework elements can be linked together. Changing one parameter in a plugin also causes a refresh of all linked parameters. We extended this software with additional plugins to enable remote user input from the mobile device.

The JPEG-encoded target (reference) images are streamed from the renderer into the tracker through a wired TCP/IP connection. In VolumeShop, a TCP server belongs to each viewport, and clients that are interested in the content change can connect to such a server. The packet format consists of a short header containing the width, height and byte-length of the image followed by the compressed frame. VolumeShop streams only when a new image is rendered. To inject interaction events into the rendering software,

an additional plugin was developed. As all the rendering properties are stored in XML format, and events occur in an asynchronous manner, we implemented a plugin that opens a UDP port and parses remote XML commands. The XML format supports different data types including integers, floats and matrices. To produce the overlays for the user, a secondary viewport was set up with own parameters and rendering properties. These parameters, for instance the modelview matrix or a transfer function value, can be linked to properties of the remote interactor plugin. Further, a new plugin was needed to convert between the different matrix notations of the renderer (row-ordered) and the tracker (column-ordered), to convert between different coordinate axes, and to convert the metrics from number of pixels to display measures. Through this plugin pipeline the user can remotely control the properties and parameters of the visualization system with the touch screen and the movements of the mobile device.

4.2 Tracker Component

A multi-purpose augmented reality framework of the Christian Doppler Laboratory for Handheld AR (CDL) has been selected as the tracker module of our augmented visualization system. A great advantage of the framework is that it enables writing portable AR applications which run both on the PC and on several mobile platforms. User applications are written in C++ and the framework can be parameterized by XML files. The core component that performs tracking is a complex computer-vision library embedded in the framework. It has several features for both marker-based and markerless tracking applications. It has been designed to support PCs as well as mobile phones with limited resources. Hence, its memory requirements are very low and processing is very fast. The theoretical background of the applied natural feature tracking algorithm is described by Wagner et al. [29], [30].

Although the CDL-tracker can perform feature tracking on several types of mobile phones, we decided to run it on a PC and stream the small camera images from the mobile device to it. Our task was to create an application that runs in the framework and exploits its natural feature tracking capabilities. The original tracker was designed to detect a static target texture on a single video input. Therefore, we extended it to allow the interactive change of our dynamic target image and recalculation of the target-feature database at runtime. During the first development stages, calibrated webcams were used to feed video into the tracker. Later, when the wireless streaming between the mobile component and the PC was working, the developers at CDL extended the framework by a network video input using the GStreamer [3] video-streaming library. GStreamer uses a filter chain to decode network streams and inject the video frames into the tracker. The cameras were calibrated using the Matlab Camera Calibration Toolbox [4] to undistort the incoming camera images for achieving better target detection. The pseudocode of the custom application within the tracking framework is listed in Algorithm 1 and 2.

In the prototype augmented visualization system, the CDL's natural feature tracker [29] is responsible for all the tracking tasks. It consist of an internal target detector and an internal patch tracker. As the detection step is computationally much more demanding, it is done once, then the tracker switches to the patch tracking mode for increased performance. For the first frame, or if the dataset has changed, i.e., a new picture arrived from the rendering component, it repeats the detection step using robust feature matching. The detection algorithm is based on a fine-tuned version of the FAST corner detector [22] [23] and the SURF [8] feature descriptor. Matching the features of the captured image with the database is done with a brute force approach in our case. As the database is not very large (approximately thousand reference vectors) and the database changes frequently, the brute force approach is not slower or even outperforms the use of any special search structure like SpillTrees, which are common in other NFT applications [29]. To compare the feature vectors, the Euclidean distance is used. After outlier-removal steps, the camera pose is estimated from the detected feature correspondences.

Once the target was found using feature matching, the internal patch-tracker component takes over and estimates the camera motion on the fly. Starting with an approximately known camera pose, e.g., from the previous frame, it searches for affinely warped versions of known features at predicted locations on subsequent images. Such a patch tracking approach is more efficient than tracking by detection. It makes use of the fact that both the scene and the camera pose change only slightly between two successive frames, and therefore, the feature positions can be successfully predicted. The patch tracker is able to track the affinely warped patches under extreme tilts close to 90 degrees and even under extreme lighting changes and reflections [30].

Algorithm 1 Pseudocode of the main thread in the custom application in the CDL-framework

```
loop
  get input video frame from the mobile device
  if PatchTracker finds previous target then
    //estimate and send pose
    estimate pose
    send pose (in XML structure)
  else
    //run feature matching
    detect keypoints on video frame
    create feature descriptors
    match descriptors to feature database
    remove outliers and calculate homography
    if target found then
      restart PatchTracker
    end if
  end if
end loop
```

Algorithm 2 Pseudocode of the thread responsible for dynamic target change in the custom application in the CDL-framework

```
loop
  //wait for new reference image
  receive target frame from network socket (blocking)
  convert color JPEG to grayscale RAW
  downsample both the old reference and the new reference
  calculate sum of pixel differences
  if sum ≥ threshold then
    //the new received reference differs from the previous one
    set Tracker inactive
    detect keypoints (for multiple scales)
    create feature descriptors (for multiple scales)
    refresh feature database (delete previous features)
    set reference as new target
    set Tracker active
  end if
end loop
```

4.3 Mobile Component

As interaction device, we selected the Google Nexus One (HTC Desire) mobile phone with the Android 2.2 operating system. The choice is based on the broad set of features (IEEE 802.11 b/g and Bluetooth connections, touch-screen, Qualcomm Snapdragon 1 GHz CPU, 512 MB DRAM, 5 MP autofocus camera, hardware support for H.263 video and JPEG image encoding, etc.) and because of the openness of the operating system. In our mobile-phone application we grab and compress the camera images. We stream them to the tracker component either as continuous video or separate frames. Further, we intercept user input and send events to the visualization system. Finally, the client is able to receive and show the VolumeShop-generated overlay imagery.

The tracker needs camera images with a resolution of $320x240$ pixels for robust pose estimation. With $8bit$ uncompressed gray values such an image takes about $320x240x1Byte = 75kB$ storage. With 25 fps this means almost $2MB/s$ transfer-bandwidth need. Without compression even the WiFi access is too slow for multiple users. With the chosen Nexus One phone, there are two favourable options to overcome this problem: either use the built-in H.263 video encoder and stream the video per RTP/RTSP or use the built-in JPEG encoder and send individual frames. We decided to avoid software video encoding (e.g., by using the open-source x264 library) because we expected high processing costs and thus significant delay on the selected phone model.

Our first attempt was to use the Android *MediaRecorder API* which accesses the camera and the hardware encoders, but it hides them from the developer. The media recorder is intended to be used for audio and video recording into files. It seamlessly compresses media and stores them on the SD-card. Android supports the MPEG4-SP, H.263, and H.264 video encoders, the AMR-NB audio encoder, and the MP4, and 3GP container formats to be used by the built-in media recorder. As the Android operating system is based on Linux, a network socket (as well as any other device) is equivalent to a file and can be accessed through a *FileDescriptor* interface. By giving a socket's file descriptor to the media recorder as destination, the compressed video output can be streamed to the network. The available video players usually support a network-stream input of standardized RTP packets. So do the VLC Media Player [5] we used for testing, and the GStreamer that has been integrated into the tracker framework. The use of RTP is desirable, so that we can avoid the manual loading and initialization of the video decoders. These steps are all done automatically in the player software. Therefore, the compressed video stream needs to be parceled into payloads of distinct RTP packets. To avoid the need of any relaying application on the PC, the RTP packets must be constructed already on the mobile device. This is solved by looping back the socket on localhost and receiving the media output in a secondary thread. In that second thread RTP packets are assembled from the received stream according to the standard [1]. To control the data flow, an RTSP server is also implemented in the mobile client. The packets are either sent to the connected RTSP clients or deleted depending on the state of an internal softswitch. The switch is set by the RTSP server thread. During testing, the VLC player could successfully connect to the RTSP server and stream the live video from the phone. However, the transfer had a significant delay of about three seconds, which is not acceptable for position tracking. The developed components could be applied in the future for other purposes, but we found that this solution is impractical for our augmented visualization system. Therefore we also implemented a second approach using JPEG-compressed frame streaming.

The tracker's GStreamer input chain is also capable of injecting video that is received as separate JPEG frames (also called Motion JPEG). The drawback of the Motion JPEG approach is that the temporal coherence between subsequent frames is not exploited for compression and thus the bandwidth need is higher. On the other hand, this method is highly suitable for our goal of having minimal delay. For the sake of simplicity, we replaced the cumbersome *MediaRecorder API* with the Android *Camera API*. The RTP/RTSP communication is also substituted by simple UDP transfer. We process small-resolution preview frames with a high refresh rate instead of taking full-resolution images. On the Nexus One, only the uncompressed NV21 (also called YUV 4:2:0) preview format with 15 fps refresh rate is available. After color conversion and JPEG compression, the frames are sent to the tracker component with delay in the order of $100ms$.

The pose from the tracker is received within a separate thread and after filtering it is also shown to the user in a text box for testing. The overlay images from the rendering component arrive into a decoder and are presented to the user. He/she can choose to see the camera preview or the overlays. Note that by extending the GUI with a *GLSurfaceView* object (a built-in Android tool to present OpenGL renderings) on top of the camera preview we could achieve traditional augmented reality.

4.4 Putting It All Together

We tested the prototype of our augmented visualization system with the following setup. The rendering component was installed on a regular desktop PC, the tracker component was set up on a commercial laptop computer, and the mobile component was run on the Nexus One phone. The computers were connected via twisted pair cables to a LinkSys WRT54GL wireless access point. The mobile phone communicated with the other components over WiFi.

Fig. 5 depicts the data flow between the individual components during operation. The operation steps are as discussed during the design section: 1) A scene is rendered and presented on the common screen and simultaneously sent to the tracker for feature extraction; 2) The mobile device captures the common scene and 3) continuously streams it to the tracker; 4) the tracker component estimates a relative pose from the two images and sends it back to the mobile device and to the rendering component to 5) enhance user interaction and to 6) render the personal overlay.

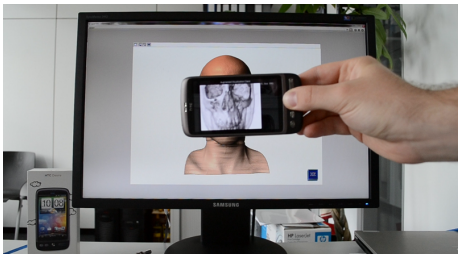
Figure 5: Prototype of our augmented visualization system. Lines with the same color represent the same data flow. Line width indicates the relative bandwidth need.

5. SYSTEM TESTS

We implemented the following test applications that should hint at the possibilities an augmented visualization system opens.

The first scenario (Fig. 6) presents a magic lens application. The volumetric data set is rendered from the calculated viewpoint of

the mobile device using a different transfer function showing other hidden features of the data (in this case the bones inside the body). The registration between the two images is not perfect due to the fact that the camera is not exactly in the middle of the phone. This is, however, a remediable inaccuracy. By applying a user-defined transformation depending on the distance between the camera and the common screen, a zooming effect becomes possible.



(a)

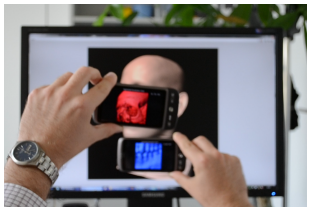
(b)

(c)

(d)

Figure 6: A medical magic lens application – both the background image and the overlay are rendered on the fly from volumetric data

The second use scenario shows a similar application with two mobile devices. At this point the multi-user setup requires multiple tracker instances on separate computers and handling of multiple video streams. However, the restriction to exactly one video input in CDL's tracker is supposed to change in the near future. As Fig. 7 shows, the clients have personalized transfer functions. Here the opacity transfer function is the same, but the colors are different. The users can move and rotate the common scene through the touch



(a)

(b)

Figure 7: Multi-user scenario with user-specific transfer functions

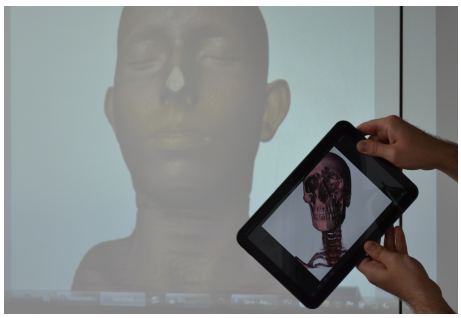


Figure 8: Orientation-dependent transfer function with a tablet

screens and trackballs of their devices.

The third test scenario (Fig. 1) demonstrates the interactive change of rendering parameters depending on the mobile device's orientation. The overlay rendering is registered to the position and the transfer function depends on the orientation of the mobile device. One can see different tissues by adjusting the orientation. The same application using a projector and a tablet PC is shown in Fig. 8.

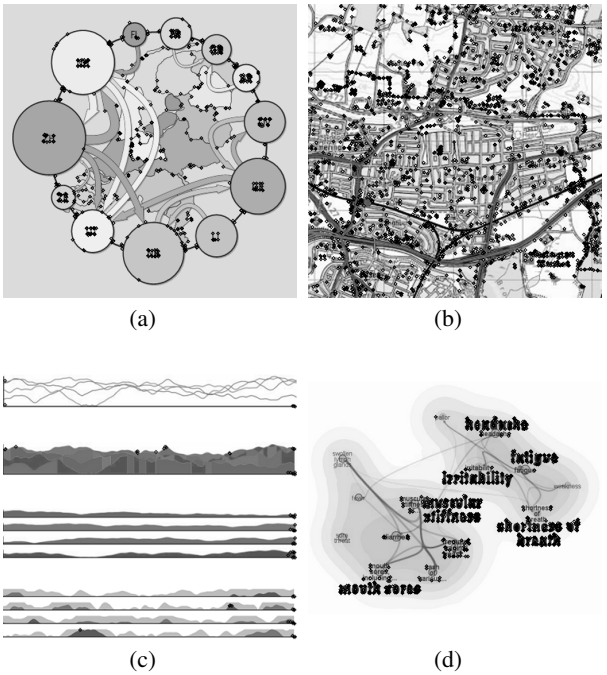
The fourth scenario is used for delay measurements. The mobile device shows the camera preview, no overlay image is generated. The estimated pose is injected into the renderer to alter the view matrix. This way the user is able to move and rotate the virtual camera in the scene by moving and rotating the mobile device around the screen. The delay between the real and the virtual movements is imperceptible.

In all applications, we achieve interactive frame rates between 8 and 20 fps. The first three applications suffer from a delay of about 0.5–1.0s between the device's movements and changes in the overlay image. The camera images are taken at a frequency of 15Hz and transferred to the tracker with a delay in the 100ms range. The fourth scenario showed that the steps up to and including pose estimation are executed very fast. Most of the delay is caused by generating the personal overlay on the PC and transferring it to the mobile device. This was the case even with very small generated images (128x128 pixels). The size of the applied volumetric dataset was 256x256x166 samples. Table 1 lists the achieved frame rates using different setup parameters. We varied the resolution and the quality of the camera images that are sent to the tracker. We found that camera images of size 320x240 pixels with JPEG quality set to 80% of the raw image are sufficient for continuous tracking. In case of a camera resolution of 176x144 pixels, the tracking degrades or fails. The abrupt change of the target image causes an outage in tracking, but a new matching step re-initializes and the system recovers in less than 1 second. The tests with a projection screen instead of a PC display involved the same calibration steps – with a small caveat: The tracking works only in the case of a distortion-

free projection, since the relative distances between feature points need to be the same as in the reference image, to successfully calculate the homography.

Table 1: Frame-rate results with the magic lens application

Camera resolution	JPEG quality	Overlay resolution	Average FPS
320x240	80%	128x128	19.2
320x240	80%	256x256	15.2
320x240	80%	320x240	14.7
320x240	80%	800x480	9.5
640x480	80%	256x256	8.8
640x480	30%	256x256	9.2
320x240	30%	256x256	16.8
176x144	80%	256x256	19.1 (degraded)
176x144	30%	256x256	failed
320x240	80%	128x128	14.2 (2 users)
320x240	80%	128x128	11.4 (3 users)



(a) (b) (c) (d)

Figure 9: Detected keypoints are indicated by black crosses on various images from IEEE Vis 2010 papers. a) Speckmann and Verbeek [27] necklace maps: 779 keypoints; b) Dykes et al. [14] map legends: 3015 keypoints; c) Javed et al. [15] time series: 17 keypoints; d) Cao et al. [13] facet atlas: 1288 keypoints

Our observation was that a reference image (1024x1024 pixels) contains thousands and a captured image (320x240 pixels) hundreds of keypoints in case of the illustrated head dataset from different viewpoints. However, the technique is not limited to volume visualization. The assumption of a relatively rarely changing, feature-rich, planar target is viable for many visualization scenarios. To provide evidence for this statement, we took several images from IEEE Vis 2010 conference papers and counted the number of identified keypoints. The images were resized to 512x512 pixels and converted to 8-bit grayscale. The applied FAST corner detector performs a test at each pixel P by examining a circle of 16 pixels surrounding P. A keypoint is detected at pixel P if the intensities of at least 12 contiguous pixels are all above or all below the intensity of P by a given threshold [22] [23]. The threshold value was set to 100, as in our previous volume rendering scenarios. The detected keypoints are shown on four sample images in Fig. 9. We found that in one out of the ten examined visualizations (Fig. 9(c)) our approach is not suitable because the number of keypoints is not high enough.

6. CONCLUSION & FUTURE WORK

The implemented augmented visualization technique contributes to previous display-interaction methods by applying markerless visual tracking for pose estimation. The presented use scenarios are achieved with interactive frame rates and acceptable delay. The prototype was built using the VolumeShop volume rendering framework, however, the technique is also suited for other visualization scenarios with any visualization software that implements the presented minimal interface. The tracking works smoothly at 15 fps, but the rendering frame rates strongly depend on the complexity of the underlying data set. The pose estimation was also tested with a large projection screen without any decrease in performance. However, a test with a glossy laptop screen with significant reflections gave negative results.

The bottleneck of the system is definitely the radio transmission in both directions. With the current implementation, the presented system does not scale well with the number of users, because the wireless bandwidth limits the number of video streams to the server. Once the tracker algorithm is available on the selected mobile platform, the system can be extended for more users with some architectural modifications as described in the following. Our application is portable from the PC to the mobile device, as it is embedded into the framework of CDL. The PC could run an instance of the application which receives the rendered images, detects the features and builds the feature database. In the next step, it sends the database to all mobile clients in a multicast manner. The clients could run our application as well, but with feature detection and patch tracking on their own camera images. This setup would reduce the required bandwidth drastically. Only the database needs to be transferred and only once to a multicast group instead of the distinct whole video streams.

Besides better scalability of the system, it would be interesting to extend our approach for multiple target images, e.g., a display wall consisting of several displays, and focus more on new interactions. The works of Jeon et al. [16] and Roman et al. [21] on interaction methods scalable for multi-user multi-display systems are good starting points. As also mentioned, the accompanying sensors of the mobile devices could be used for hybrid tracking with improved accuracy. The accelerometer data could be utilized to turn the mobile phone into a Wiimote-like 3D interaction device. The mobile user interface could be further enhanced by a bundle of features, for instance a context-sensitive GUI for each participant, etc. A built-in algorithm for automatic calibration of the transformation matrices for different displays would make our system an out-of-the-box enhancement for visualization systems. In conclusion, our prototype implementation opens a wide variety of research directions.

7. ACKNOWLEDGMENTS

The authors thank the members of the Christian Doppler Laboratory for Handheld AR for assisting with the experiments. This work has been partially funded by the Austrian Science Fund (FWF), grant no. P21695 (ViMaL).

8. REFERENCES

[1] RFC 4629: RTP Payload Format for ITU-T Rec. H.263 Video. www.ietf.org/rfc/rfc4629.txt.

[2] ARToolKit, (last checked on 20.08.2011). www.hitl.washington.edu/artoolkit.

[3] GStreamer, (last checked on 20.08.2011). www.gstreamer.freedesktop.org.

[4] Matlab Camera Calibration Toolbox, (last checked on 20.08.2011). www.vision.caltech.edu/bouguetj/calib_doc/index.html.

[5] VLC Media Player, (last checked on 20.08.2011). www.videolan.org/vlc/.

[6] R. Adelmann. Mobile phone based interaction with everyday products - on the go. In *Proceedings of the 2007 International Conference on Next Generation Mobile Applications, Services and Technologies*, pages 63–69, Washington, DC, USA, 2007. IEEE Computer Society.

[7] R. Ballagas and M. Rohs. Mobile phones as pointing devices. In *Pervasive Mobile Interaction Devices (PERMID 2005), Workshop at Pervasive 2005*, pages 27–30, 2005.

[8] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110:346–359, June 2008.

[9] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 73–80, New York, NY, USA, 1993. ACM.

[10] S. Boring, M. Altendorfer, G. Broll, O. Hilliges, and A. Butz. Shoot & copy: phonecam-based information transfer from public displays onto mobile phones. In *Proceedings of the 4th International Conference on Mobile Technology, Applications, and Systems and the 1st International Symposium on Computer Human Interaction in Mobile Technology*, Mobility '07, pages 24–31, New York, NY, USA, 2007. ACM.

[11] S. Boring, D. Baur, A. Butz, S. Gustafson, and P. Baudisch. Touchprojector: mobile interaction through video. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems*, CHI '10, pages 2287–2296, New York, NY, USA, 2010. ACM.

[12] S. Bruckner and E. Gröller. Volumeshop: an interactive system for direct volume illustration. In *Visualization, 2005. VIS 05. IEEE*, pages 671–678, 2005.

[13] N. Cao, J. Sun, Y.-R. Lin, D. Gotz, S. Liu, and H. Qu. Facetatlas: Multifaceted visualization for rich text corpora. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1172–1181, 2010.

[14] J. Dykes, J. Wood, and A. Slingsby. Rethinking map legends with visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):890–899, 2010.

[15] W. Javed, B. McDonnel, and N. Elmqvist. Graphical perception of multiple time series. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):927–934, 2010.

[16] S. Jeon, J. Hwang, G. J. Kim, and M. Billinghurst. Interaction with large ubiquitous displays using camera-equipped mobile phones. *Personal Ubiquitous Computing*, 14:83–94, February 2010.

[17] H. Jiang, E. Ofek, N. Moraveji, and Y. Shi. Directpointer: direct manipulation for large-display interaction using handheld cameras. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 1107–1110, New York, NY, USA, 2006. ACM.

[18] D. Kalkofen, E. Mendez, and D. Schmalstieg. Interactive focus and context visualization for augmented reality. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR '07, pages 1–10, Washington, DC, USA, 2007. IEEE Computer Society.

[19] T. Quack, H. Bay, and L. Van Gool. Object recognition for the internet of things. In *Proceedings of the 1st International Conference on Internet of Things*, IOT'08, pages 230–246, Berlin, Heidelberg, 2008. Springer-Verlag.

[20] M. Rohs, J. Schöning, A. Krüger, and B. Hecht. Towards real-time markerless tracking of magic lenses on paper maps. In *Adjunct Proceedings of the 5th Intl. Conference on Pervasive Computing (Pervasive), Late Breaking Results*, pages 69–72, 2007.

[21] P. Roman, M. Lazarov, and A. Majumder. A scalable distributed paradigm for multi-user interaction with tiled rear projection display walls. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1623–1632, 2010.

[22] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1511, October 2005.

[23] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.

[24] J. Sanneblad and L. E. Holmquist. Ubiquitous graphics. In *ACM SIGGRAPH 2005 Emerging technologies*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[25] H. Slay, M. Phillips, R. Vernik, and B. Thomas. Interaction modes for augmented reality visualization. In *Proceedings of the 2001 Asia-Pacific Symposium on Information Visualisation - Volume 9*, APVis '01, pages 71–75, Darlinghurst, Australia, Australia, 2001. Australian Computer Society, Inc.

[26] H. Slay and B. Thomas. Interaction and visualisation across multiple displays in ubiquitous computing environments. In *Proceedings of the 4th International Conference on Computer graphics, Virtual Reality, Visualisation and Interaction in Africa*, AFRIGRAPH '06, pages 75–84, New York, NY, USA, 2006. ACM.

[27] B. Speckmann and K. Verbeek. Necklace maps. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):881–889, 2010.

[28] M. Spindler, S. Stellmach, and R. Dachselt. Paperlens: advanced magic lens interaction above the tabletop. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, pages 69–76, New York, NY, USA, 2009. ACM.

[29] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Real-time detection and tracking for augmented reality on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):355–368, 2010.

[30] D. Wagner, D. Schmalstieg, and H. Bischof. Multiple target detection and tracking with guaranteed framerates on mobile phones. In *Proceedings of the 2009 8th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '09, pages 57–64, Washington, DC, USA, 2009. IEEE Computer Society.