論文の内容の要旨

論文題目　Projected Reality - Augmenting the Environment with a Network of Controllable Video Projectors -

(プロジェクテッド･リアリティー -複数のビデオプロジェクタを協調利用した環境の拡張現実感に関する研究-)

氏名 エナス ヨハン

A lot of research on Augmented Reality (AR) has been done aiming to support people doing complex tasks. The main idea is that computer generated views of virtual objects are overlaid onto the view of the real world. These virtual objects could mark the position of a button to press, show the correct position a lever of a machine should be in, or show where certain parts need to be inserted next. The possibilities seem to be endless. However, AR still has not made the step out of the laboratory into wide spread applications. I believe that this is mainly due to the way AR has been implemented so far. Most AR systems use Head Mounted Displays (HMD) to overlay the virtual objects over the real world. While HMD certainly have been improved since their invention by Sutherland in the late 1960s, they are still kind of cumbersome and heavy to wear, or their field of view is too small to augment the whole environment.
Besides these physical reasons, also social reasons speak against their usage: They block eye contacts, a very important part of non-verbal communication. This isolates the users wearing them from other people in the same room. Another fact that is often ignored by developers of AR systems is that they make their wearer appear like a ‘geek’. While this does not matter for the ‘typical AR developer’, it creates a resistance within the intended users to accept the new technology. Since this happens on a subconscious level, it is very hard to counter the resulting rejection.

Consequently it has become my goal to develop a system that uses video projectors in a very flexible way to be able to augment objects while users may move them around. Since the range of a single projector is limited and projectors can only project onto surfaces that face them, it is necessary to have several projection units, each controlled by a computers, in a room. A software architecture had to be developed that enables these projection units to communicate with each other in order to find the best suited projection system to augment surfaces in real time. This architecture should scale well up to global scales with many thousands of projection systems while being powerful enough to allow for complex interactions involving several projection systems in a certain area. Furthermore this architecture should hide all the networking/roaming functionality from the developer of AR applications, which run on this AR systems like conventional applications run on an operating system. Ideally the application developers should not even think about the fact that their applications may be using several projection units at runtime. The involvement of additional projection units shall happen completely transparently and automatically.

The first milestone to reach that goal was to develop a single prototype of an AR projection system. This AR projection system is based on the pan and tilt able projector AV4 from Active Vision. Not only the pan and tilt of the projector can be controlled from the computer, but also zoom, focus and several other projector parameters. While the projection of the augmentation certainly is an important functionality, it is equally important to know the exact position and orientation of the objects that shall be augmented. Video tracking was the obvious choice, as it can be integrated into the projection system well and has similar limitations as the projection system. Both work just within a certain range and the projection surface has to be visible from the point of view of the projector and the camera. To provide this functionality I mounted a camera on top of the projector so that the unit always 'sees' what is in front of its projector. The AR-ToolKit Tracking library is used to detect markers in the captured video frames and to calculate their position and orientation towards the camera. This information can be evaluated further to control the projector's orientation and focus. While this functionality is sufficient to project simple graphics onto an object, the behavior was hard coded in the first prototype.

The next step consequently was to develop an Application Programming Interface (API) in order to make the development of applications easier. Only this way more complex behaviors became possible. This API is based on my concept of Projected Applications. The basic idea of Projected Applications (PA) is as follows:

The AR-projection units themselves have no knowledge about how to augment different objects. This information is coded as 'Projected Applications', which are analogous to applications in a GUI environment. However, while conventional applications interact with the user via windows and widgets on a computer's screen, projected applications use tracked objects for interaction. The AR-Projection systems provide means to identify tracked objects as well as to measure their positions and orientations. They also project the output onto these tracked objects or other fixed and known surfaces, such as walls (coded in the projected application). The AR-system can be seen as an operating system that loads and executes projected applications and provides an abstraction layer for these applications to communicate with the user and to control the hardware.

Based on that API the system can be extended to use several projection units. For this third milestone an application server was introduced to store and manage the projected applications. However, the application server not only serves the applications for the AR-projection systems. More importantly, it also maintains the state of the projected applications. It therefore has to ensure that the state is only modified by one AR-system at the time, which means that only one projection unit may execute an application and augment the corresponding object at any given moment.

Once a projection unit detects a marker in the image captured by its camera, it sends the ID of the marker to the application server. In reply to that, the application server sends the application and, if available, the display rights and state of the application back to the projection unit. Now the projection system starts the new application. If the unit was granted the display rights and sent the last state of the application, it initializes the application with its state and starts to project the augmentations. If the display rights were not be available at that moment, the system does nothing but to try to follow the object and wait to be granted the display rights.

If a system that owns the display rights for an application cannot detect the relevant

marker any longer, it returns the display rights together with the current state (encoded in a container object) to the application server. It can reapply for the display rights once it detects the marker again.
In the mean time, the application server may send the state as well as the display rights to another projection system.
While this simple method of managing the display rights requires only minimal amounts of network bandwidth and is sufficient if the ranges of the projection systems don't overlap, it is not satisfactory if more than one system could perform the augmentation at the same time. In that case the system that detected the marker first gets the display rights and keeps them until the marker disappears from its camera's view. However, it would be better if the system with the best view of the object would project the augmentation. Furthermore, the management of the display rights had to be more dynamic and find a new projection system that takes over before the active one looses the object and the augmentation disappears completely.
Consequently the next milestone was to extend the management of the display rights and make it more dynamic. The application server can actively withdraw display rights (in combination with the applications' states) from a projection system now. This way it can give the display rights (and states) to better suited projection systems at any time, long before the augmentations disappear completely on the systems that held the display rights before.
However, in order to decide which system gets the task to augment certain objects, the application server needs to know which system is suited best to perform the job, which may depend on the three main criteria: distance, direction of the surface normal and a free line of sight. The quality can be considered to be a weighted combination of these criteria. However, the weighting may be quite task- or application specific.
In order to keep the task of the application server simple, I introduced a scalar quality value. The system with the highest quality value is considered the best and consequently should perform the augmentation. Since the criteria of quality can be very task specific, it is up to the application developer to define the function to calculate this quality value. However, a default function should work reasonably well in most cases.
The AR projection systems regularly send updates of the quality values of all applications they host to the application server.
In consequence the application server can easily compare the quality values of the different applications running on the available AR projection systems and in turn can ensure that every application runs on the optimal projection system.
I performed a user study that compared different functions to calculate a quality value with the way human test subjects would evaluate the quality of different projections onto a test object at different distances and at different angles towards the projector.
As a result I found a function that better matches the dependence of projection quality dependent on distance and orientation according to human perception.
While the system so far worked quite well for projected applications using only one tracked object and augmenting only one surface of it, it became clear that the restriction to project only using the projection unit the PA is running on is a severe limitation. Many objects of daily life are more like a box with several surfaces than a board with only a front side. While it is possible to project onto several surfaces of an object using one projection unit, it is only possible as long as they all face this projection unit. In order to

augment an object from different sides it has to be possible to augment this object using different projection units at the same time, with each surface being augmented by the unit that fits best. In order to make that possible while still keeping the applications' states consistent, I developed the concept of Hydra Applications. Analog to the creature from ancient Greek mythology which had one body and many heads, the projected application runs on only one unit, the first one that requested the application from the application server, and any further units requesting the application will get a reference to that first unit. The additional units send their tracking information to the executing system and in return they get information about what to project onto which surface. Based on the Model-View-Controller design pattern, the communication between the different units is hidden in the Hydra Controller, which is part of the system. Application developers only have to implement the model, the behavior of the application that is executed on only one system, and the view objects, which represent the different surfaces that the application can augment. Besides enabling the augmentation of several surfaces of an object, this approach also makes it possible to augment several objects that may be in the range of different projection units by a single application. An other possibility introduced by this feature is that users may wear stereo glasses tracked by one system while another one would project stereoscopic images with the correct perspective for the users' eyes onto surfaces, creating the impression of a three dimensional augmentation.