Prototype application

Ubiquitous Communications (UbiCom) Program
From project P2: Visual Information Processing and Applications
Application deliverable
Status: approved by P2 (23 January 1998).
distribution: internal (P1, P2 and P3).

Wouter Pasman
Arjen van der Schaaf

What to do with this report?

Before **18 February** P2 requests feedback on the following points:

- Do the proposed applications cover your project? Does it inspire and challenge you, and will it give the answers that you need? What is lacking?
- Can your project be mapped over the proposed partitionings that are given in the Appendix? Are there discrepancies between our partitioning and your project?
- Do you see problems with the proposed applications or partitioning that are not resolvable within a year?

Please email to W.Pasman@twi.tudelft.nl or Arjen@it.et.tudelft.nl.

Application proposal

In this report we propose an application for the first prototype of the UbiCom system. We aim at an application that is simple but yet sufficiently inspiring, allowing to make appropriate choices and to get an appealing whole. We propose to develop a toy application for test purposes, followed by a serious application.

In Table 1 we attempted to assign a 'degree of inspiration' to a number of applications. For our rating we did not consider application independent parts (rendering and position estimation by means of camera images, as discussed in the Appendix). We rated the applications with regard to the following aspects:

- Backbone: whether the application has a large number of jobs requiring the backbone (except rendering and position estimation).
- Video captures: to what extend the application uses camera images recorded in the headset (except position estimation and z-map construction).
- Multi-location: whether the application runs in different places (e.g., cells of the RF transmitters).
- Transmission errors and bandwidth: to what extend does the quality of the RF channel play a role.
- Occlusion: whether virtual objects have to disappear behind real objects.
- Blocker: whether real objects have to disappear behind virtual objects.
- Alignment: to what extend is accurate positioning of virtual objects required.
- Stereo: whether binocular disparity is important for the application.
- Lag: to what extend is the application sensitive for lags in the image refresh.
- Static (not moving) virtual objects: does the application use them?
- Moving virtual objects: does the application use them?
- Video: the extend to which the user wants to view remote video images.
- Interaction with the system: the amount of information the user sends to the system. The user can interact (explicitly or implicitly) with the system either by moving, by pushing buttons or by handling menus.
- Interaction with other users: to what extend does this occur?
- GIS: the intensity with which the application uses a GIS database (except position estimation and the construction of the simplified virtual world).

Table 1. Degree of inspiration for several parameters (+ = inspiring, o = neutral, - = not inspiring). The applications are put vertically, the different aspects horizontally. Toy applications are listed at the top, followed by a selection of more serious applications.

	Backbone	Camera recording	Multi -location	Transmission errors, bandwidth	occlusion	blocker	alignment	stereo	lag	static virtual objects	dynamic virtual objects	video	interaction with system	Interaction with other users	GIS
Pacman	-	-	-	+	+	+	-	+	+	+	+	o	o	+	-
adventure	+	o	+	+	+	+	o	+	+	+	o	+	+	++	+
paintball	+	+	+	+	+	+	+	+	++	o	o	o	o	++	+
tourist info	+	-	++	-	-	-	-	-	-	o	o	+	+	-	++
library info	+	-	+	-	-	-	-	-	-	+	-	-	+	o	++
salesman	+	-	+	-	-	-	-	-	-	-	-	o	o	-	o
family doctor	+	+	++	-	-	-	-	-	o	-	-	+	+	+	o
surgical support	++	++	-	+	+	+	++	++	+	+	-	o	o	++	-
barber	-	o	-	-	+	+	++	++	o	+	-	-	-	-	-
military	++	++	++	++	+	-	+	+	++	+	o	+	+	++	++
security	+	++	++	+	-	-	-	-	+	o	o	+	+	+	+
police	++	++	++	++	o	-	o	o	++	o	o	+	+	++	+
architect	+	-	+	+	+	+	+	+	+	++	-	o	o	o	++
electrician	+	-	++	o	-	o	+	++	+	+	-	o	o	o	+
geographer, GIS	++	++	++	o	o	o	++	o	o	+	-	-	+	-	++

Summarising Table 1, we conclude that the toy applications are quite inspiring, although not as much as some of the serious applications on certain aspects. The overall most inspiring serious applications seem to be surveillance-like (military, security, police) or GIS-based (architect, electrician, geographer). The surgical support application does have many plusses in Table 1 but it is not very well suited for the UbiCom project, because it lacks mobility.

However, the 'degree of inspiration' is only part of the story: a first application also has to be testable, simple, modular and scalable (Table 2). With testable we mean that we can sufficiently control the set-up to do precise measurements, for example, in a conditioned environment. With simple we refer to the number of lines of code and number of hardware components required. With modular and scalable we mean that a simple version of the application can be extended to form a more complete, more interesting or more appealing system.

We hope to build a working prototype within one year. From Table 2 it is concluded that the toy applications are best suited for such a first prototype. The pacman application seems the most simple and best testable, since it is confined to a well-defined and controllable labyrinth. Among the serious applications, the GIS database application seems more simple and testable than surveillance-like applications. Developing a military application may also raise ethical objections from UbiCom group members. Due to camera recording, privacy-issues may pose problems to all surveillance-like applications.

Table 2. testability, simplicity and scalability of the application (see text)

Application	testability	simplicity	modularity and scalability
Pacman	++	++	+
adventure	+	+	+
paintball	+	+	++
tourist info	+	o	++
library info	+	+	+
salesman	-	o	++
doctor	-	o	++
surgical support	-	-	o
barber	o	+	-
military	-	-	+
security	-	o	+
police	o	-	++
architect	o	o	o
electrician	o	o	++
geographer, GIS	+	+	++

Conclusion

As a toy application for test purposes we propose a labyrinth application (Pacman and Paintball) as these are easy to implement and testable in a conditioned environment. The application is easy as the task is well-defined and requires limited interaction.

However, these applications do not cover all aspects of the UbiCom project and do not support serious tasks. Therefore other applications have to be developed, such as a GIS database oriented application. One possibility is an architectural application: a design has been made to adapt the university campus, for example a tram, parking place or a new building. The architect and his employer must be able to view the proposals by walking through the campus. This application will be worked out later. Also, a library information application may be set up, using the new university library to test GIS-like principles.

Possible cooperation

It may be possible to connect to the project 'GIS and VR', a project investigating visualisation and planning of the Dutch railways. Furthermore, cooperation may be possible with Riek Bakker, who is actually working on a 'stedenbouwkundig masterplan voor the TU' (town-planning master plan for Delft University) (Delta 29, nr. 30).

Other possibilities for cooperation exist with the DIOC-9 MISIT project (Minimally Invasive Surgery and Interventional Techniques, see http://www-mr.wbmt.tudelft.nl/~dankel/dioc9.html). This project may be relevant mainly for the surgical support application. However, drawbacks of this application are its very restricted environment (is 'wireless' needed and allowed in a surgical environment?) and the non-use of a GIS.

Partitioning of the system

A number of aspects of the partitioning of the system that are independent of the application are described in the Appendix. The application dependent partitioning is discussed below, with the applications.

Pacman application

Description of the application

Before the game starts the player must stand on a well-defined position. That position can be indicated, for example, with a virtual marker. As yet this application is thought indoors. The room is augmented with virtual objects, which form a labyrinth in combination with the real objects in the room. Next, the player can walk through the labyrinth. Probably the player has a map of the labyrinth, possibly supplemented with his own position. If the player walks through a (virtual) obstruction, he has to return to the place where he went through the obstruction before he can continue playing. Finally, multiple players may walk through the labyrinth simultaneously.

There are virtual pills in the labyrinth, that can be eaten by walking over/through them. When all pills are eaten the player is finished and may continue to the next level.

There are also ghosts walking through the labyrinth: –possibly transparent– blobs that can catch the player. New ghosts enter the labyrinth at a fixed place. A player that is catched must restart his game on the same labyrinth. There are a few special pills in the labyrinth. When the player eats one of them the roles of player and ghosts is reversed temporarily (10 seconds?): now the player can catch ghosts. A catched ghost disappears from the game, and is replaced by a new ghost after a while.

Functions that the application consists of

• Seen from the player:

A player must be able to start the game, walk to the initial position, walk through the labyrinth, get pills, see real and virtual objects, catch ghosts/get catched, stop the game, look at his map, and possibly look at the camera image of an other player.

• Seen from the software:

Generation of a (GIS) database describing the room, generation of labyrinth and pills, directing the player(s) to the initial position, tracking of the position of the player(s), simulation of movements of the ghosts, detection of collisions between player, pills, objects and ghosts, display updates to maintain the virtual world and to show camera images of other players, redirect the player to the last legal position when he made 'illegal' movements.

Application dependent partitioning of the system

Simulation of ghost movements can be calculated on the backbone. Collision detection probably can be done most easily in the headset, the handling of it in the backbone. The precise implementation of collision handling also depends on the choice for the architecture (P3) and the processing power in the headset (P1).

Benchmark options
The player has to be able to walk fast without disturbing rendering errors in the display. A total lag of 30 ms seems acceptable for this task.

Ranges to be investigated (approximately in order of importance):
- Problems due to lag (from 30 ms to 100 ms; with occluding walls possibly 10 ms).
- Necessity of a blocker to remove real-world objects, and to create shadows (Off-line simulation may show that blocking is not required for Pacman).
- Necessity of occlusion (do virtual objects have to disappear behind real objects? The ghosts do not have to occlude real-world objects. The labyrinth will be more exciting when occluding objects can be placed, because this lowers the overview on the labyrinth.
- Necessity of stereoscopic images (do cheap hacks such as disparity per-image instead of per-pixel work?)
- Constraints with respect to alignment (is position tracking with cm-accuracy sufficient?) (test range 5 cm - 20 cm)
- Rendering and warping performance: when all walls in the labyrinth are virtual, the system requires high rendering capacity. However, the UbiCom system was supposed to add some small things to the real world, and thus this may be undesirable. Another possibility is to build a real labyrinth, blocking may be less relevant in this case.
- Compression rate (range to be filled in)
- Bit error rates (range to be filled in)
- Bandwidth (range to be filled in)
- System architecture (several scenarios, e.g. tasks on the headset versus on the backbone)

Modular construction of application
Application independent modules were discussed in the appendix. We can distinguish the following modules and their relation in the pacman application.
'==>' means: requires.

```
pacman application
==>  labyrinth database
     visualisation of labyrinth (see below)
     ghosts
     ==>  ghost route planning
          ghost visualisation
          ==>  rendering and displaying
     pills that pacman can eat
     ==>  visualisation of pills
          detection that player eats pills (localisation)
          response on eating pills
          ==>  visualisation (remove pill) and probably some sound
               reaction of ghosts
     rules of the game
     ==>  detection whether player collides with walls, ghosts, etc.
```

visualisation of labyrinth
==> initialise
rendering
==> viewpoint tracking
rendering techniques
transmission
==> communication channel
compression
displaying
==> see-through display & blocking
image stabilisation for movements of observer
display control

Options for implementation

Many of the required hardware parts (see Appendix) may be first implemented with off-the-shelf parts or may be ignored or bypassed at early stages of development and added later. For example, position tracking may be done at first with an off-the-shelf magnetic tracker; the retinal scanning display may at fist be substituted by off-the-shelf LCD or CRT displays; image warping may be ignored at first; wireless transmission may at first be bypassed, or simulated with wired transmission, or implemented with an off-the-shelf mobile telephone; data compression may be ignored at first or implemented with standard techniques.

Several steps in the software development can be distinguished:

implement position and orientation tracking
implement image stabilisation by image warping
implement displaying static objects through rendering
display the labyrinth from a data base
add data compression
add network protocols
add mobile link (or simulation)
add user interaction
add multiple users
add more functionality

A major question is: at which stage will independent parts be integrated. We do not necessarily have to integrate all parts at the same time. For example: display and image stabilisation may be brought together at an early stage, while the mobile link, for example, may be added at a later stage, or, it may be simulated. The steps from first experiment to final prototype that are to be taken are grouped as

off-line / desktop	to	real-time
local	to	mobile
software	to	hardware

We suggest the following stages in the prototype development:

1: off-line desktop simulation in software
2: hybrid phase (combined off-line and real-time components)
3: real-time mobile implementation in hardware

The hybrid phase is necessary to facilitate early tests and to avoid that the project must wait for the slowest developing parts. This stage is meant to test the system and not to have a working application.

To investigate whether a blocker (see Appendix Figure 5) is required we will do off-line simulation. Furthermore, we intend to build simple prototype blocking hardware to test whether blocking is feasible.

Benchmarks layout

The pacman application must be well playable. This application sets fair but not extreme requirements for the system as to occlusion, blocker, stereo, and lag. The moving ghosts may impose a problem if the system is designed exclusively for static objects. In that case, the problem may be circumvented by making the ghosts jump instead of move.

The tasks have minor requirements for communication with the backbone, usage of the cameras, and the GIS. This makes this application less interesting for these aspects of the system. However, extra possibilities for testing may be created with supplementary tasks, or by extending the labyrinth functionality to support the paintball application.

Paintball application

Description

A labyrinth like the one in the pacman application may be used, but this application may also be played in the open. Multiple players are now required. The players hold a virtual gun which they can use to virtually shoot other players. Instead of the pills that are found in the pacman application, the players can now find virtual ammunition. Ghosts are excluded from the game. Extra options, like placing booby-traps, may be added to the game at a later stage of development.

A paintball game has one of several goals. (1) Players must defend their flag and steal the flag of other players (by walking over it). (2) Players must eliminate other players from the game with one or more hits. The last surviving player wins. (3) Fox hunt: several players (hunters) chase a single player (the fox), who tries to hide or escape. Points are rewarded to the fox for the time he survives.

Functions of which the application consists

- From the players point of view:
  Players must be able to start the game; walk around; pick up ammunition, see real and virtual objects; shoot other players, or be shot by other players; communicate with other players and share their camera views.
- From the software's point of view:
  The system needs to generate a (GIS) database that describes the environment; place virtual objects; track the position of players and their virtual gun; detect collisions between players and virtual objects; detect hits; update the displayed information to show current environment or camera views of other players. In case of a player making an illegal movement (such as walking through a virtual wall) the system must direct him back to his original position.

Partitioning the system

See the Appendix for an application independent partitioning. Application specific partitioning contains (at least) collision detection (including hits), which is 'user interaction' and must be handled with short lag times. Collision detection may be processed at the headset or at the backbone (where the database with all player positions is), or distributed over both. It may need a dedicated data transfer channel with short lag-times, or a direct broadcast to other players.

Modular construction

paintball
==> labyrinth-database
labyrinth-visualisation (see the pacman application)
multiple users
==> multiple headsets
multi-user support
weapons
==> user to system interaction
response to user interaction
==> visualisation of hits and misses
rendering and displaying
rules
==> detection of collisions between players and virtual objects
detection of hits en misses

Benchmark options

Ranges to investigate: see the Pacman application, but now smaller tolerances are to be expected.

- Lags (10-40 ms)
- Higher need for blocker and occlusion
- Alignment (1 cm-5 cm)

Benchmarks

Action is the key to this game, not: waiting for the system to respond. Lags will probably be quickly annoying in this application. Probably lags as short as 10 ms will already be perceivable. This application is a severe test for a blocker. Errors in the blocking function will result in severe problems when a player tries to hide behind a virtual object. This is also a 'killer application' for the z-map generator. For example, a player hides behind a virtual wall or door and peeks with one eye through a pin-hole to shoot other players. In this situation, it will be extremely difficult to generate a z-map from two displaced camera views. Summarising: this application requires the most of each part of the system.

Appendix: partitioning of the UbiCom system

The general UbiCom system layout is shown in Figure 1. A proposal for more detailed partitioning is given in this Appendix.

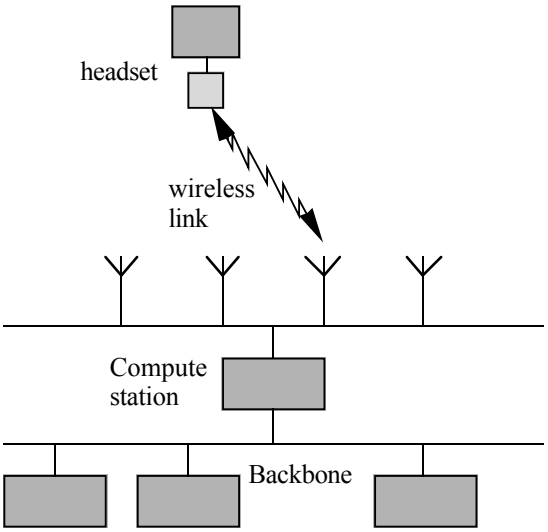

Figure 1. General system set-up (after Figure 2 from 'Ubiquitous communications: program proposal to ARTD').

A number of requirements related to lag, alignment, stereo, occlusion, and blocking were proposed in an earlier report ('Perceptual requirements and proposals for the UbiCom augmented reality display', Pasman, 1997). Lag is the time between request (e.g., following head motion) and the corresponding update of the display. The earlier report states 10 ms as a requirement for the maximum lag in the display stabilisation process. This is necessary for head-mounted displays, as they are moved through space while the displayed information must be stable and seemingly attached to the real world. Alignment denotes the accuracy at which virtual objects are positioned in the real world. Occlusion means disappearance of virtual objects behind real objects. Blocking means disappearance of real objects behind virtual objects.

We propose the system partitioning as given in Figure 2 to achieve a display update with a maximum lag of 10 ms. This Figure concerns a distributed form of rendering. In the first step, the virtual world in the GIS database is simplified to a few 3D polygons and images (textures) that are relevant for the current position of the observer. Crudely estimated, this step may take a bit longer than 500 ms. The 3D polygons and textures are rendered to a single 2D image or a few sprites (images of independent objects) in the second step. This step may take between 50 and 500 ms. The rendered image (or sprites) are corrected with a fast image warp technique to the actual viewing position and head orientation in the third step, just before displaying. This step must reach the required maximum lag of 10 ms.

A number of variations on the scheme of Figure 2 are possible. The first variation concerns the place of the wireless link: it can be placed between step 1 and 2, and between step 2 and 3. In the first case, polygons and images (textures) are sent over the wireless link, and more processing power is required in the headset. In the second case less processing power is required in the headset, but higher performance is required for the wireless link that has to transport large amounts of image data

with short lags. The second variation concerns the display stabilisation: it can be done based on sprites or based on a single image. Additionally, both sprite- and single-image-based stabilisation can be implemented in different ways, for example rotation-only, rotation- and translation combined or full image warping. The third variation concerns step 2, where the polygons and textures can be static or dynamic (and this choice has consequences for step 3). For example, the textures can be video-images and polygons can move through space.

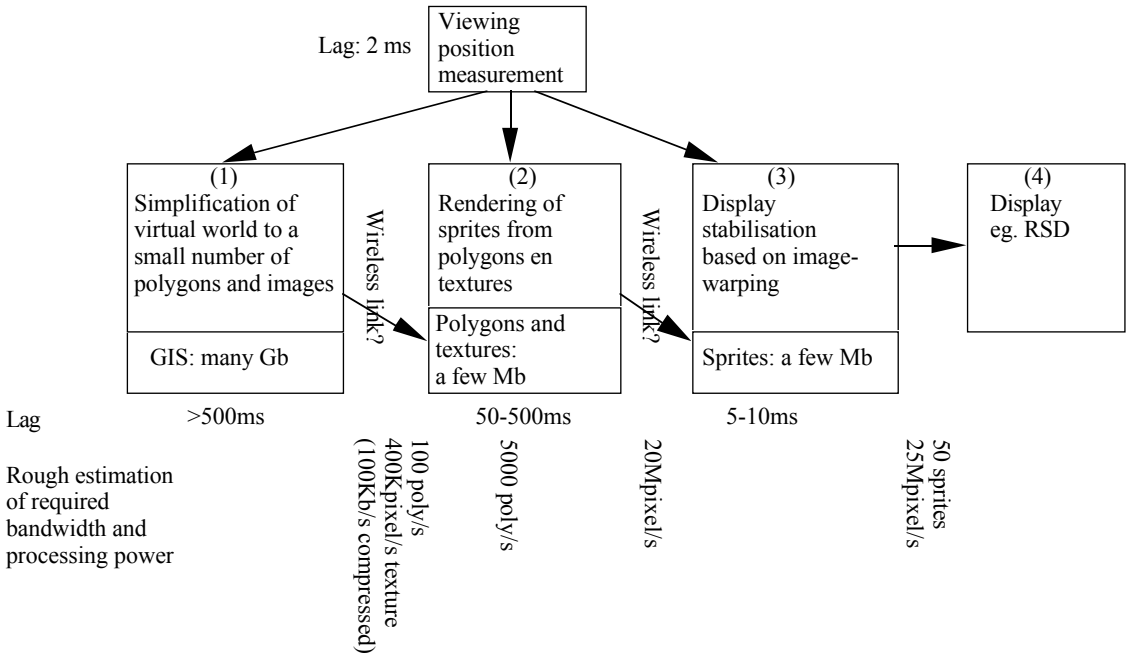

Figure 2. Possible system set-up for distributed rendering. In this set-up the wireless link can be placed at different locations. See text.

Figure 3 and 4 show two extreme alternatives in more detail. Figure 3 shows a more detailed example of the processes in the headset and in the backbone. Here we assume a wireless link between step 1 and 2 in Figure 2. The left part of Figure 3 shows the processes in the headset, the right part processes in the compute station and backbone. The top part shows processes that are required for the position- and orientation tracking of the observer, the bottom part the processes that are required for the image generation.

The tracking is based on a combination of inertial tracking (gyroscope and acceleration meters) and visual tracking. Possibly a global positioning system or antenna information is required to determine the initial position. The inertial tracker is very fast and is the basis of the tracking. Errors in the inertial tracker are small but are accumulating, and therefore a periodic correction is required. The images from the camera are time-stamped and send to the backbone. There the objects in the camera image are recognised and compared with the data in the GIS. From this, the recent camera position is determined. This recent position is sent back to the headset. Simple objects can be tracked also in the headset. The position measurements from the backbone and the headset are combined to correct the inertial tracker.

The recent position is also used in the backbone to construct a simplified virtual world. This simplified world is sent to the headset to be rendered to sprites. In Figure 3 we chose to use sprites that can be rotated, translated and scaled for the image stabilisation.

Depth information from the environment (a z-map) is required to get the occlusion and blocking right. Such a z-map can be generated in real-time with special hardware and two camera images with

a fixed disparity. The display and blocker (that blocks out light from the real world) can be steered based on the depth information from the real and virtual world. When a virtual object moves behind a real object the light from the display has to be blocked or not generated (occlusion); when a real object moves behind a virtual object the light from the real object has to be blocked. Blocking and occlusion are included in Figure 3, but are not essential for many applications.

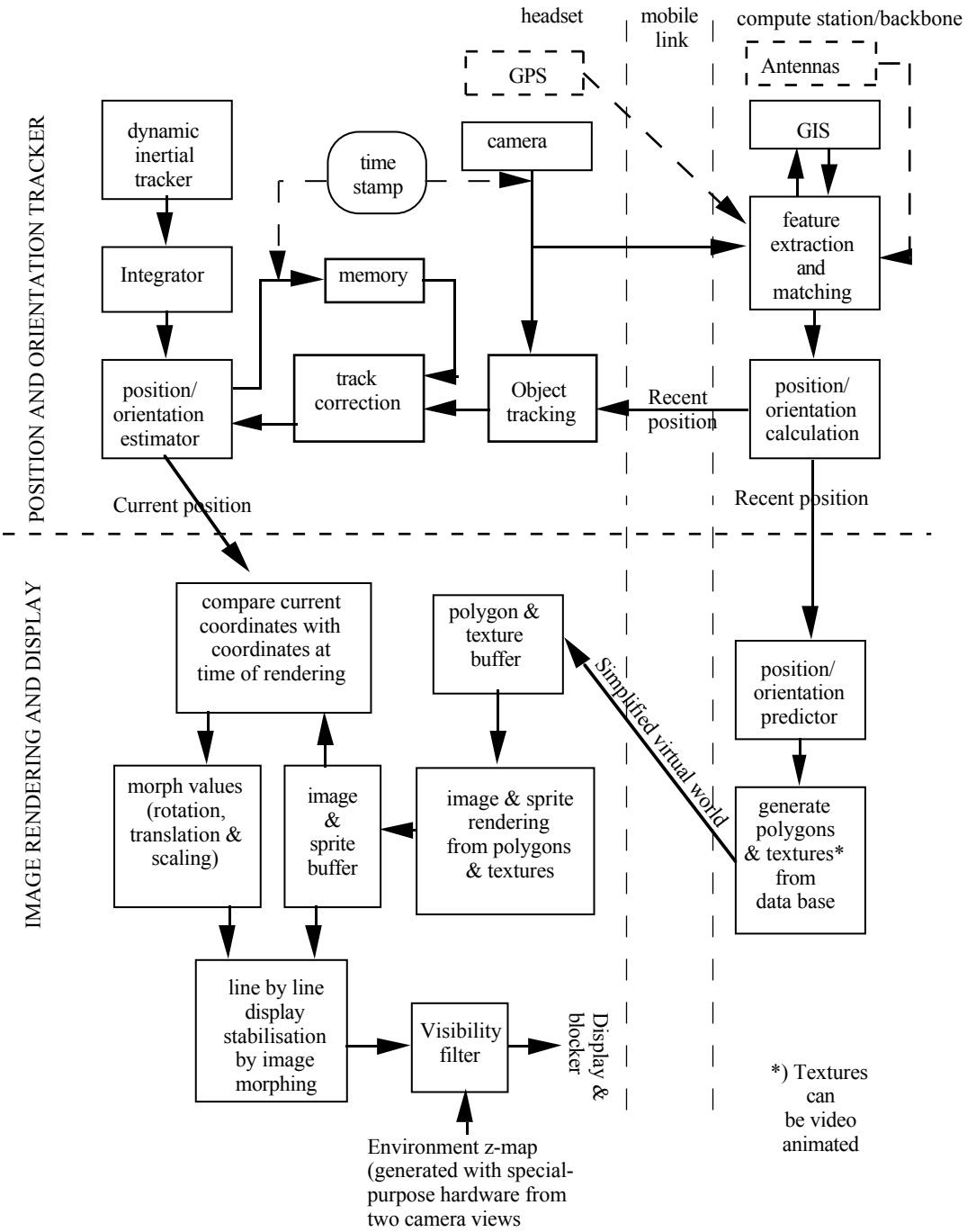

Figure 3. More detailed partitioning of rendering and tracking processes. See text.

Figure 4 shows an alternative detailed partitioning of rendering and tracking processes, contrasting with Figure 3. Here, the visual tracking is done entirely on the backbone. A compass and inclinometer are used in the headset instead of an inertial tracker. These give only orientation

measurements. Here the backbone renders images instead of simplifying the virtual world, and the image stabilisation concerns only head rotation. No environment z-map is generated and occlusion and blocking are absent. In this set-up, the backbone and mobile link are heavily loaded, while the processing capacity on the headset is minimised.

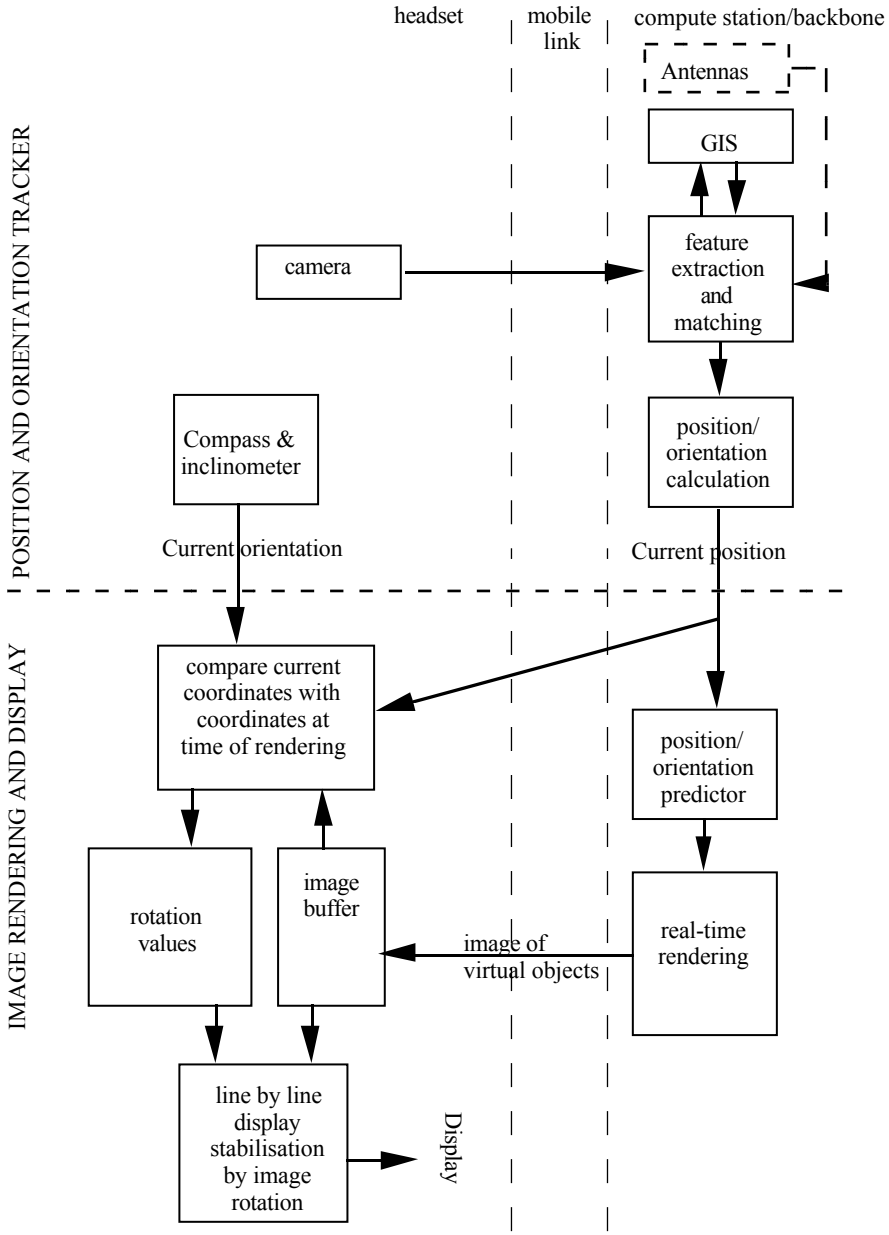

Figure 4. Another possible detailed partitioning of rendering and position tracking processes. See text.

Figure 5 shows a possible implementation of the display and blocker processes in more detail. This example concerns a retinal scanning display with blocker, but other imaging formation techniques can be used. In that case the pupil tracking and image beam guidance may be less essential. As pupil movements may be very fast, dedicated hardware in the camera may be used for pupil tracking. In this example, the pupil tracking guides the image beam into the eye. In more sophisticated setups it may also serve attention directed rendering or as a mouse pointer. In Figure 4 no blocker was used, thus the associated processes in Figure 5 can be omitted in that case.

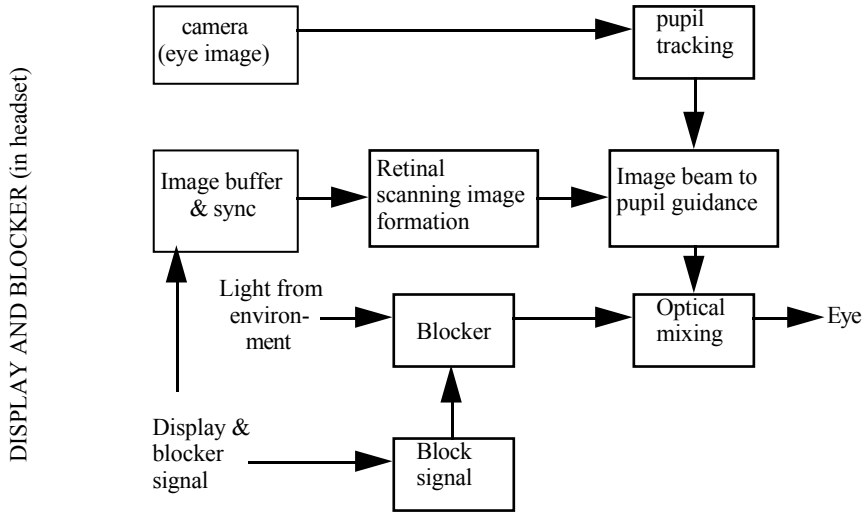

Figure 5. Detailed display and blocker processes.

Scalability

Figures 3 and 4 show two quite extreme situations that require either high processing power in the headset or high bandwidth of the mobile link. Other partitionings may be developed to trade processing power for bandwidth on a sliding scale. The UbiCom system may be designed to support scalable versions of the headset: cheap headsets that require high bandwidth and expensive headsets that have more processing power and are more robust against degraded transmission. A modular approach to this concept is to support an optional, portable compute station (like a laptop) that the user may carry and connect to his headset. However, this scalability-scheme concerns only the trade-off between processing power and bandwidth, but not hardware components like gyroscope, blocker and fast image warping hardware.