Warping for image stabilisation

Discussion report

W.Pasman, 25/2/98

Introduction and overview

With a see-through augmented reality (*AR*) display the image in the display should be updated within a few milliseconds after a head movement of the observer, in order to maintain the virtual objects at a stable location in the visible world. In other words, the virtual camera has to mimic the head movements of the observer as close as possible. Allowed lags are in the order of 10 ms (Pasman, 1997; Azuma and Bishop, 1994).

These constraints are difficult to meet, even with existing very fast polygon rendering engines. Optimised polygon rendering on high-performance Silicon Graphics machines still gives total lags of about 30 ms (Jacoby, Adelstein, and Ellis, 1996; Keran, Smith, Koehler and Mathison, 1994). And additionally, in our wireless AR display we should also minimise the required computing power.

To solve these problems, a previously rendered image can be deformed (*warped*) to approximate a full rerendering at the new camera position. Appendix A describes different types of warping. As warping may be done 10-20 times faster than a complete rerendering (Torborg and Kajiya, 1996), this may resolve the lag problems. Ideally the warp is controlled by control points in 3D space, in order to find a warp that brings the image as close as possible to the completely rendered image. But approximations may be used to avoid costly perspective projections.

To generate warped images that mimic the effect of a camera rotation, or to *simulate camera rotation* (Figure 1), the rendered image has to be shifted and/or rotated as a whole. To simulate camera translations a more complex transformation is required: parts in the image depicting more distant objects have to shift relatively less than objects closer to the camera. Therefore, simulating a camera translation requires more sophisticated techniques than simulating camera rotation.

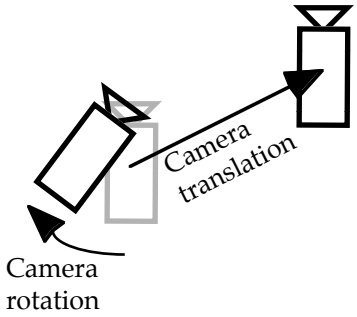


Figure 1. Any camera movement can be decomposed into a camera rotation and a camera translation.

The image to be warped can be devided in several depth layers. Each layer can be warped differently and then merged to form a single image. Such separate parts are called *layers* or *sprites*. This approach allows better simulation of camera translations (see below).

Warping seems to have two advantages over standard polygon rendering. First, a speedup of up to 20 times (Torborg and Kajiya, 1996) may be reached from warping as compared to polygon rerendering. By using warping several expensive division operations that are required for perspective projection are exchanged for cheaper approximating multiplications. Nevertheless, I am curious about this speedup, as similar approximations may be used for polygon projection, and texture mapping takes most of the computational resources both with warping and with polygon rendering (see Appendix B). Second, some warping types allow us to use optimised scanline-algorithms instead of per-polygon actions. Such an algorithm allows minimisation of the end-to-end lag by calculating the next pixel that will be displayed just before it is needed (Figure 2). Of course this only works if the warping parameters can be calculated fast given the current viewing position.

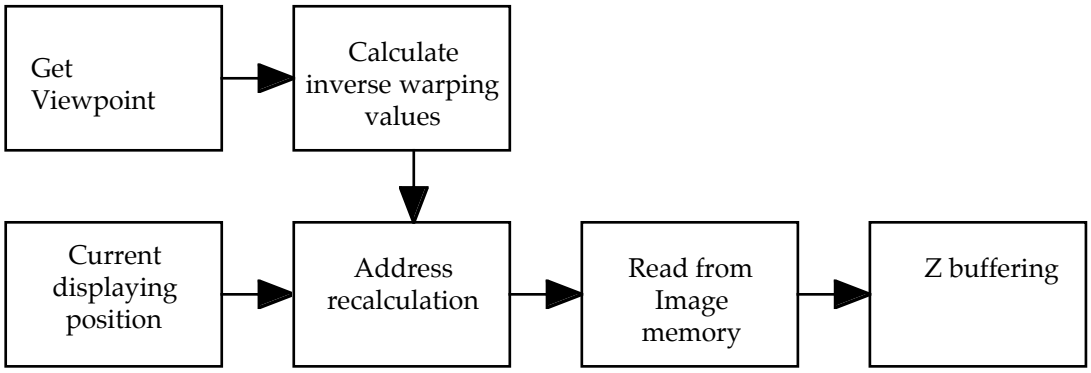


Figure 2. General layout of an address recalculation warping system.

A warp is only an approximation of a complete rerendering, and it is not clear to what extend the approximation suffices and what warp type is appropriate. This report discusses the possible solutions, their consequences and proposes tests that have to be done to estimate what warping is appropriate for the first prototype system. First, I will sum up the results from the earlier report (Pasman, 1997) concerning the requirements of the system. Second, I will discuss the different ways pictures and (partial) depth information can be represented, and its consequences. Third, technical aspects and requirements for fast warping are discussed. Fourth, existing real-time warping systems are reviewed, and their usefulness for our purposes is discussed. These systems were not designed for stabilising AR images, and have drawbacks when used for our purposes. The fifth section discusses ideas for improvement and an overview is presented of the findings. Finally, a selection is made of the most promising systems that should be investigated in more detail.

Requirements

In earlier reports (Pasman, 1997; Pasman & van der Schaaf, 1998) it was suggested that for AR the angular precision should be below 0.5˚. Furthermore, we want to

provide stereoscopic images to the observer. A blocker that can block light from the environment in order to substitute objects in the real world complete with virtual objects seems highly desirable. Finally, we expect that lags of 0.5 to 1 second will occur in the connection link between the backbone and the headset.

As we want an optical merging of virtual-world and real-world images, the total system lag is an important parameter. In 0.5 to 1 s, the observer can easily make full 180˚ turns. Furthermore, if we assume he is walking with 1 to 2 meters per second, we can expect translations of up to 2 meters. Our stabilisation mechanism should be able to cope with these lags and displacements.

Lags may also be avoided by predicting the viewpoint. However, the required precision and the amount of lag are too high to stabilise AR images. For example, Azuma and Bishop (1994) found that at a prediction interval of 100 ms the average error is 0.36 cm and the peak error is 15 cm. Mark, McMillan and Bishop (1997) indicate that the predicted error grows with the square of the lag, so at 0.5 s lag we can expect average errors of 9 cm and peak errors of 3.75m.

Picture representation and consequences

This section discusses the choices that have to be made regarding the representation of the pictures and their consequences. First, pictures can be planar or surrounding. Second, complete or partial depth information may be available to steer the warping process. Third, monoscopic or stereoscopic images may be available.

Planar versus surrounding image

The warps described in Appendix A work with planar source images. As such a planar image always covers less than 180˚, required parts of the image will lack when a camera rotation has to be simulated - e.g. when the observer rotates his head. We will call lacking image parts in the target image *visibility gaps*. Figure 3 illustrates an example of a visibility gap. This problem can be solved with surrounding images. Figure 4 shows three examples of surrounding images.

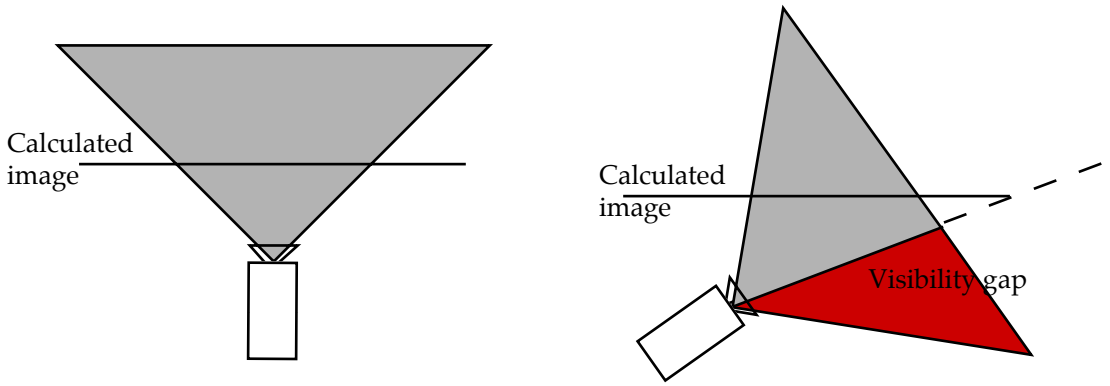


Figure 3. When using planar source images, required parts of the target image may lack (visibility gap) when simulating camera transformations, especially rotations.

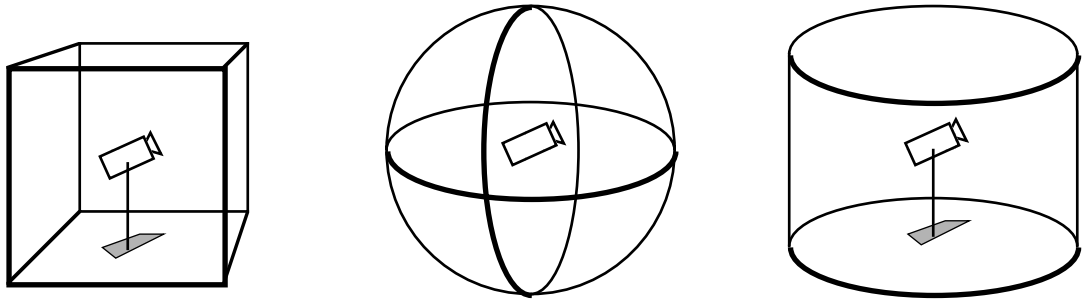


Figure 4. Examples of surrounding images. In (a) several planar images are combined to form a surrounding cube. In (b) the source image is spherical. In (c) the source image is cylindrical.

A surrounding cube (Figure 4a) has some advantages. Regan and Pose (1994) summarise:

> "The surface of a cube was chosen as the encapsulating rendering surface after considering many possible candidates, as the cube results in a moderately simple hardware implementation and has fewer aliasing artefacts than most other surfaces. The cube's greatest advantage however is in its associated rendering simplicity. The surface of the cube may be rendered to by rendering to six standard viewport mappings." (Regan and Pose, 1994, p. 156)

However, the system of Regan and Pose can simulate only camera rotations. When simulating camera translations, the boundaries of the faces may pose problems to warping. This is because images of objects crossing an edge of the cube will be split over different faces and warped separately. As the separate warps may contain different approximation errors, the images may not connect neatly at the edges. Hirose, Watanabe and Endo (1998) used a large video database with both cylindrical and cubical surrounding images (this is not in their paper but they told about it) to allow real-time virtual walkthroughs. They found that cylindrical modelling was too coarse for warping, and preferred cubical surrounding images.

Such boundaries are avoided by using a surrounding sphere (Figure 4b). Using polar coordinates does not require the polygon renderer to use polar coordinates too, but instead an image warp can be done to bring an image into its polar version.

On the other hand, polar coordinates may also cause problems with warping, as rotation of sprites around an axis out of the image is not straightforward in polar coordinates. Furthermore, straight lines in the 3D scene will not be projected to straight lines in polar space (as was the case when projected to a flat plane). For example, Figure 5 shows the checkerboard of Figure A3 when placed at a distance of a quarter of its width from the camera and transformed to polar space. This will give troubles with the warping, as running through linear space is easier than running through a curved space (see the discussion on scanline algorithms under 'real time warping').

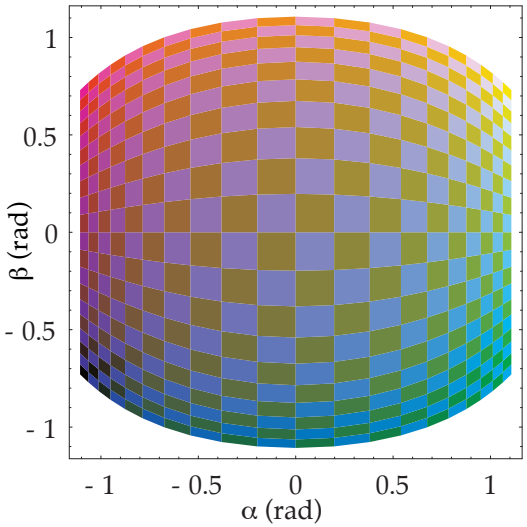


Figure 5. The checkerboard of Figure A3 put into polar form.

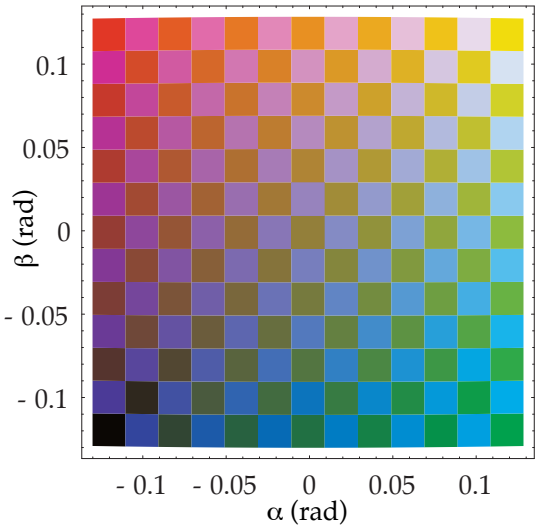


Figure 6. The centre of the figure is nearly square. Here the central 30˚ by 30˚ is shown

One way to make rotation easy while still using a polar representation is to approximate areas on the sphere with a normal flat rectangle. The errors at the boundaries of the rectangle will be small if the area is small. It can be seen in Figure 6 that the squares in the centre are nearly rectangular. Figure 7 shows for various distances from the centre how close the approximation is to perfectly square. The quality at 0.1 rad (11.5˚) is 99.7%. For example, let's assume a display resolution of 640 pixels spread over 30˚. A sprite reaching up to 0.1 rad from the middle (thus having a width of 23˚) will have an error of 0.3% or (0.3% * 23/30 * 320) = less then 1 pixel at the borders.

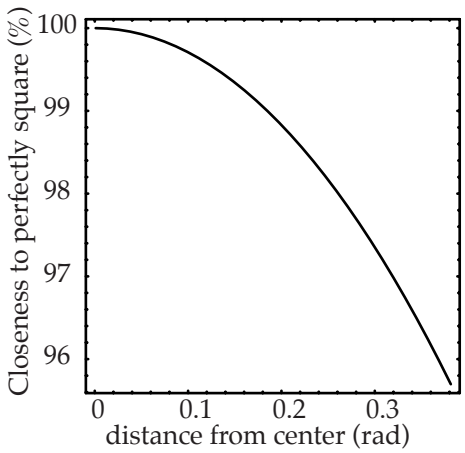
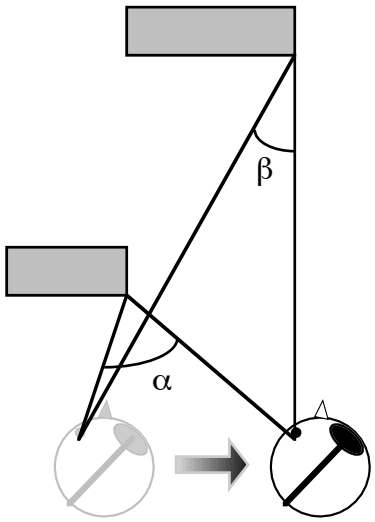


Figure 7. Closeness of the approximation to perfectly square as a function of the distance from the centre.

Cylindrical coordinates (Figure 4c) are also possible (Apple, 1998a) and have the advantage that the viewing space can be captured easier into a single image than polar coordinates. However, cylindrical coordinates do not cover the whole environment. For our purposes inclusion of at least the floor seems important.

Partial depth information

Surrounding images allow perfect simulation of camera rotation. But to simulate camera translations, depicted objects should shift and scale proportional to the length of the translation and inverse proportional to their distance to the viewpoint (Figure 8). Thus, to simulate a camera translation, we need the distance between the viewpoint and the objects in the picture. This section discusses the possibilities to include depth information in the images and problems when insufficient depth information is available.



Translation parallel to object

Figure 8a. An observer translation parallel to objects results in a visual shift of objects that is inversely proportional to the distance between the objects and the observer.

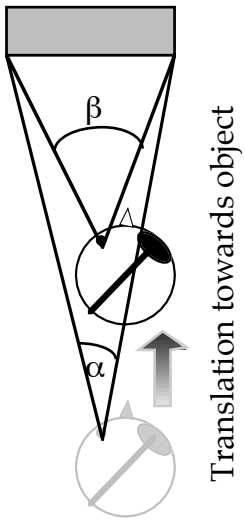


Figure 8b. Similarly, an observer translation towards an object results in visual scaling of the objects, again inversely proportional to the distance between the objects and the observer.

We have a problem with camera translations when using a single image without depth values (Figure 9). In such a case, a warp can approximate camera translations only very roughly, assuming some distance between the observer and the image. For the observer, the virtual scene will look like a picture instead of giving him the suggestion of space. Consequently, misalignments with objects in the real world will be unavoidable in the AR situation.

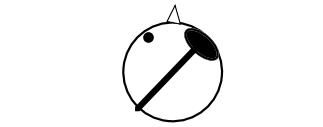


Figure 9. with a single image some distance has to be assumed to approximate camera translations

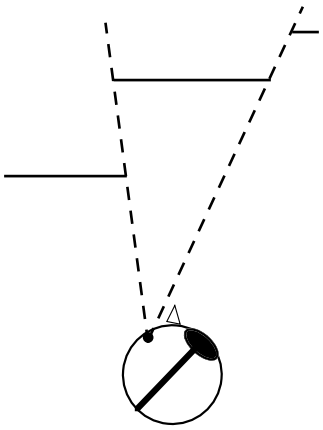


Figure 10a. depth values assigned to parts of the image.

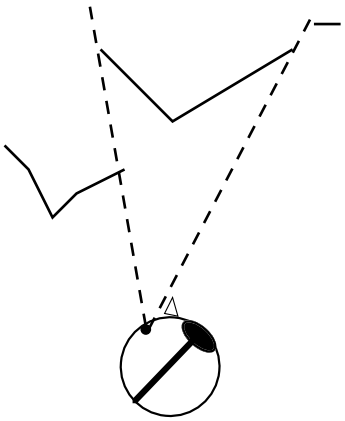


Figure 10b. Instead of approximating the scene with flat images, a polygon structure may be used.

A first step to improve warping possibilities is to assign a depth value to parts of the image (Figure 10). However, this does not completely solve the problems with camera translations. In this situation, of objects hidden behind other objects only part of the image is available. When the camera translates new parts of the hidden objects should become visible. As these new parts are not available in the old image, this again results in lacking image data: a visibility gap. Figures 11 and 12 illustrates a visibility gap occurring in the museum walk through simulation of Mann and Cohen-Or (1997).

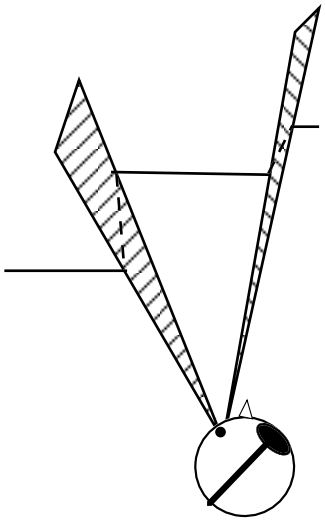


Figure 11. this approach gives 'visibility gap' problems with camera translations.

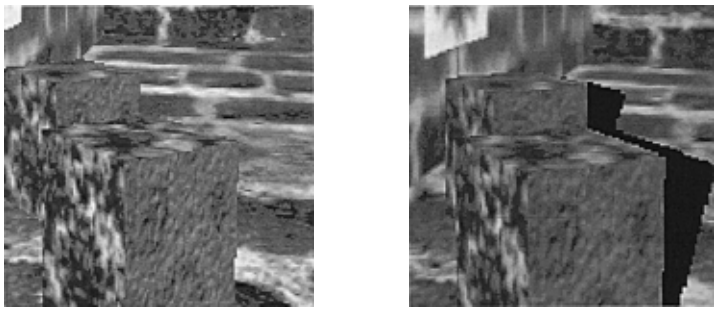


Figure 12. Left: A single image to be warped. Right: when the camera translates (here to the right) some parts of the target image are not visible in the source image. These parts are black here (From Mann and Cohen-Or, 1997).

A straightforward way to solve this visibility gap problem is to store extra information at the depth discontinuities to anticipate observer translations (Figure 13a). As a translation disoccludes more of close objects than it disoccludes distant objects, less anticipating image data is required for more distant occlusions. But the problem with the visibility gap still may occur with extreme observer translations. A less straightforward way is to use not a single but

multiple source images (Figure 13b). Picture information lacking in one image may be available in other frames.

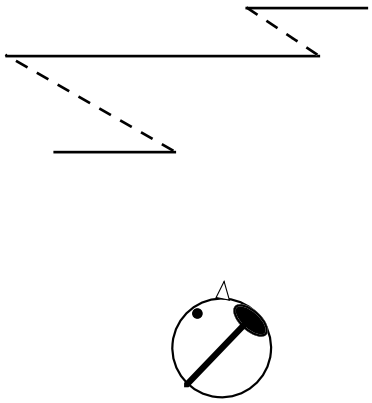


Figure 13a. Extra information added to anticipate translations. For occlusions at further distances less anticipating data is required.

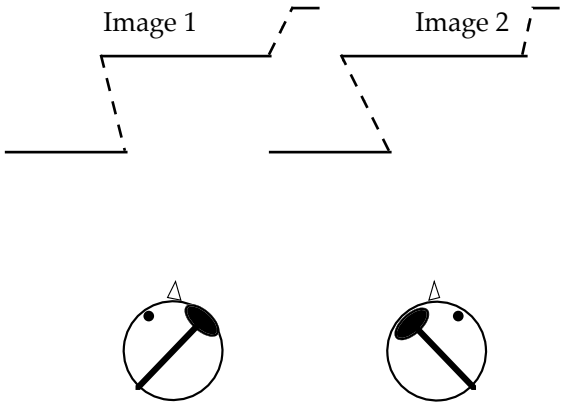


Figure 13b. A less straightforward (and redundant) way to store the extra information is to use multiple source images.

A more natural approach seems to use a separate image for each object (*sprites*) (Figure 14). This allows per-object warping at an appropriate resolution and refresh rate. Such selective rerendering also allows speedup of the polygon rendering (eg., Shade, Lischinski, Salesin, DeRose and Snyder, 1996) and straightforward handling of video images that may be placed on an object. Furthermore, with sprites (partly or semi-) transparent objects can be handled conveniently.

In some situations it may be necessary to model the objects even closer than with a single image. The object may be split into several images, each to be warped independently (Figure 15). This situation is similar to standard polygon rendering, both in scene representation and in required computing power (see Appendix B), but the warping approach may offer better solutions to the lag problem than standard rendering.

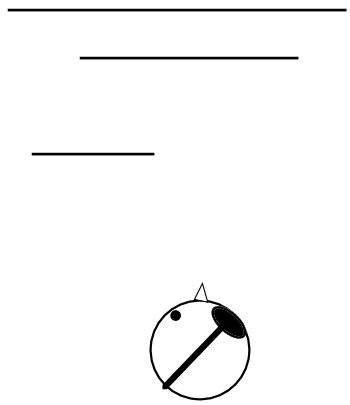


Figure 14. Multiple overlapping pictures perpendicular to the viewer's line of sight. Occluded parts might be available in lower resolution.

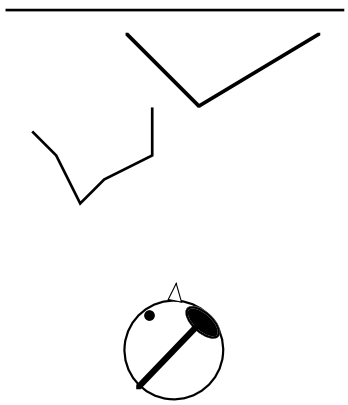


Figure 15. Multiple pictures for each object, approximating the object's surface. Closer objects are more detailed.

Stereoscopic images

The requirement to generate stereoscopic images may easily double the required processing power, but with some tricks only a fraction of the processing power may suffice to attain a reasonable stereoscopic pair.

Combining surrounding images with stereoscopic images may be a problem. Consider two surrounding images, one for each eye. If we just take the two surrounding images from the position of the left and right eye of the observer (Figure 16a) and the observer rotates his head 90° along the vertical axis, the disparity will be zero (Figure 16b).

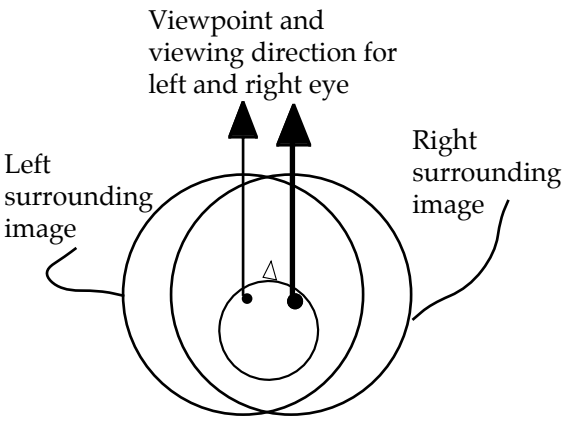


Figure 16a. Problem with stereo combined with surrounding images. Assume that centres of surrounding images correspond to eye position in some situation.

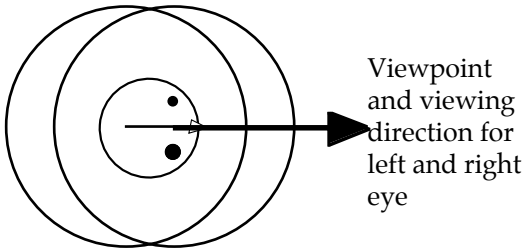


Figure 16b. If the observer rotates his head (here 90°), these centres are incorrect. Consequently, the images have incorrect disparity.

Musgrave (1992) proposed to make special surrounding images, by making a surrounding image using a centre of rotation not coinciding with the camera lenses (Figure 17). This may be easily done with ray tracing systems, but for real time rendering this approach is not feasible.

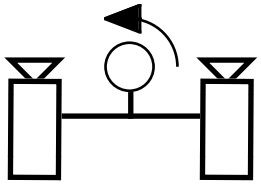


Figure 17. The problem shown in Figure 15 may be attacked by using special surrounding images, generated by rotating the camera off-axis. Such images are easy to generate with ray tracers, but not with polygon renderers.

Alternatively, the depth information in a normal pair of stereoscopic images can be used to extract depth information for the warping process. Next, this depth information might be used to warp the images. However this depth extraction requires much computation power and leaves us with a representation close to Figure 9, giving problems with visibility gaps in the warped images.

A more efficient approach is to have image data and some depth information at a single viewpoint, and to warp this image to get the image for the other eye. The

situation and problems in this case are similar to the general warping problem: given a view from some eye position and depth information, we want to generate a view from the other eye position in a cheap and fast way. Shifting just the sprites of the objects in the image pairs such that their disparity is approximately right is expected to suffice, as disparity is a fast but not very strong depth cue. I expect that visibility gaps are highly disturbing for stereoscopy, and therefore representations like Figure 9 seem less suited.

Concluding, it seems appropriate generate a stereoscopic pair with the same warping mechanism as required to for the image stabilisation. As stereoscopy essentially requires a translation of the camera, the warping system has to be able to simulate at least horizontal translations without visibility gaps.

Conclusion

A surrounding image seems essential to cope with observer head rotations which may be very fast. For stereoscopic images, the system should be able to simulate horizontal translations without visibility gaps.

Using the sides of a surrounding cube for a surrounding image may give address recalculation possibilities that are easy to implement. However, this system seems less flexible with regard to per-object rendering and less efficient with image storage. Considering the complications that may arise when generating stereoscopic images, using sprites in a polar coordinate system seems the appropriate choice. A brute-force polygon rendering approach may be feasible but nearly doubles the required rendering power to generate stereoscopic images as compared to monocular images.

Technical aspects of fast warping

There are several strategies to implement a warping system. The warping system may copy each source pixel into the appropriate location in the target image (*forward mapping*). Alternatively the warping system may run through the target image and find for each pixel in the target image its location in the source image (backward mapping or *address recalculation*). Of course address recalculation is only possible if an inverse warping function exists.

The advantage of the address recalculation system over the forward warping system is that it can handle any image scaling in a straightforward way. With a forward warping system upscaling of the image is especially tricky, as source pixels should be replicated or interpolated. A failure to do so will cause holes in the target image (Figure 18).

Figure 18a. Example of errors caused by forward warping, when simple 1-1 pixel copying is used. Image to be warped.

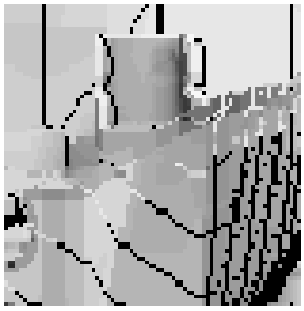


Figure 18b. Warped version of Figure 18a. Simple pixel copying will cause holes in the upscaled target image. (From Mark, McMillan and Bishop, 1997).

Efficient versions of address recalculation often are *scanline algorithms*. Scanline algorithms decompose the 2D warp into a series of 1D warps. This allows simpler antialiasing schemes and more efficient memory access than general address recalculation. The scanline algorithms that can be mapped effectively on hardware usually consist of an iteration of a simple function to approximate the exact warp. For example, if the pixels are located in memory on locations base_address+x dy/dx (where dy/dx is given and x runs from the lower to the highest required value with increments of 1) then the subsequent pixel addresses may also be attained by starting with current_address:=base_address and repeating the loop current_address:=current_address+dy/dx. This saves one expensive multiplication per pixel (multiplication roughly takes 20 times the computing power required for addition, eg., Morris, 1994). For perspective warps, such a linear approximation is not sufficient, but a quadratic function is a good approximation (see Appendix A) while all multiplications still can be replaced with additions.

More complex warps can also be done with scanline algorithms, but they require several in-between images and an equal number of scanline warps (alternating in the horizontal and vertical direction). Wolberg (1990) describes these algorithms in detail. Because this buffer will introduce high lags in the warp, such an approach is not useful for our purposes.

If a polygon renderer is available in the headset, this may relax the requirements for the rest of the warping. We can assume a typical value of 100 ms for rendering and an additional 100 ms before the next rendering becomes available, e.g. a maximum lag of 200 ms. In 200 ms, the observer will be able to make rotations of about 20˚ (probably 40˚, but at such high rotational speeds I expect that he will not notice errors in the display) and translations of up to 40 cm.

If the polygon renderer re-renders objects instead of whole scenes, the lag may be lower for certain objects. The object coming into view will have a longer lag at the start of this movement than when all rendering power is distributed evenly over all objects, but once it enters the centre of vision the object will be rerendered quickly. Therefore per-object rendering can expected to lower the latency that has to be compensated for by the warping.

A related problem that can be solved with scanline algorithms is the high memory bandwidth required for video streams. A standard video image already requires 25 Hz x 640 x 480 x 3 bytes (per pixel) = 23 Mb/s. Furthermore, twice this bandwidth (giving a total bandwidth of about 50 Mb/s) seems necessary as the

video streams also have to be written into the memory with the same speed. To reach the full power of a retinal scanning display with brute force, say a stereoscopic 4000x4000 at 100 Hz, an enormous bandwidth of more than 10 Gb/s is required. A standard 70 ns RAM has a throughput of 14 Mb/s and clearly is insufficient. Special RAMs exist such as VRAM, FBRAM, SDRAM and RAMBUS. VRAM (Foley, van Dam, Feiner & Hughes, 1987) and FBRAM (Deering, Schlapp & Lavelle, 1994) were designed especially for video systems. FBRAM is optimized as z-buffer and has a bandwidth of 400 Mb/s. SDRAM promises to reach peak rates of 528 Mb/s (Hitachi, 1998), and RAMBUS chips allow 600 Mb/s or more when contiguous blocks of about 2K are requested (OKI Semiconductor, 1998). However, rotation of images requires quite irregular memory access.

It is possible to get extremely regular memory reads if the warping process is split into two stages: the first stage simulates translations while the second stage simulates rotations. The translation simulation would require only scaling and translation of sprites, and therefore sprites are accessed always in the same (left-right, top-down) order. This stage results in a single image with only the closest visible pixels. The subsequent rotation stage only has to rotate this single image, thus avoiding the highly irregular memory access of the direct warping approach. However, this two-stage approach requires a full frame buffer between the first and second stage, which requires memory to store a full frame, and worse, will give lag times of a full frame.

Existing warping systems

This section discusses the work that already has been done on real-time image warping, and what solutions might suffice for our purposes.

Translation-only hardware

A system that might be adapted for warping purposes is the MPEG (de)compression system. For MPEG (de)compression, special purpose hardware capable of real-time translation of parts of images is available. Therefore, the image is cut into blocks of 8x8 or 16x16 pixels. A few frames are compressed as a stand-alone image (I-frames) in a JPEG-like manner. Other frames are compressed by using image parts from earlier I-frames and correcting data (P-frames). Most frames are compressed using image parts both from the preceding frames (up to the latest I- or P-frame) and future P-frames (B-frames). Figure 19 shows the compression dependencies. Because the displacement of reused blocks depends on the motion of the objects in the scene, the displacement vectors of reused image blocks are called *motion vectors*. These motion vectors might also be used for warping of the image. Using the B-frames of the MPEG system would cause unacceptable delays, but B-frames need not be used. But then, it still remains unclear whether full frames are being buffered.

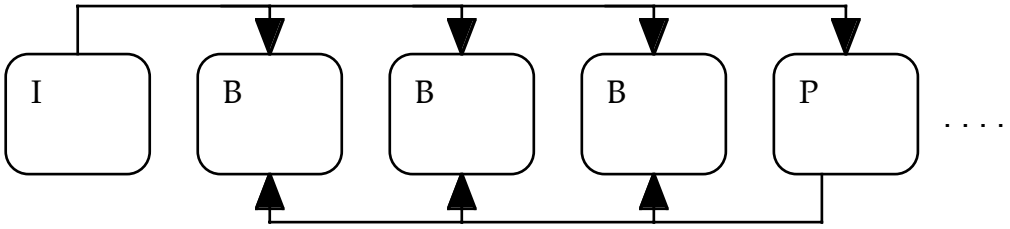


Figure 19. Compression dependencies in MPEG. See text.

In high definition television (HDTV), a coding scheme similar to MPEG is used. The objects and motion vectors are transmitted separately. At the receiver site, the motion of objects is interpolated from the motion vectors as long as no new motion vector is received.

MPEG is also the basis for another high-quality television system: HD-MAC. HD-MAC has a 100 Hz refresh rate, but not all parts of the image are refreshed with 100 Hz. Displacement of image parts based on the motion vectors is used to interpolate images. In contrast with HDTV, the object determination and the motion estimation are both done at in the receiver (Fernando, Parker & Rogers, 1990).

Special hardware exists both for the determination of the objects and motion vectors (Harrand, Mougeat & Perron, 1992) and for motion interpolation (Fernando, Parker & Rogers, 1990). The determination of objects and motion vectors from video frames seems less interesting for our project, as more sophisticated motion estimation is possible because the 3D object motion is known.

A problem with these MPEG-like systems is that it works with flat images, usually exactly the area to be displayed, which causes visibility gap problems as described in Figure 3 and 10. Furthermore, the absence of scaling possibilities can not be tolerated to simulate camera translations of up to 2 meters.

Rotation-only hardware

A first system that can do mesh warps is the system proposed by Wolberg (1990). His system consists of a two-pass warp (Figure 20) where a single image can be warped arbitrarily in real time, including fold-overs. The warp is defined by placing a grid over the image and indicating where these grid points should be warped to (see Figure A8). In the first pass, the image is warped only vertically (Figure 20a). In the second pass, the warp is completed by doing a horizontal warp of the half-warped intermediate image (Figure 20b). For some parts in the image the horizontal warp should be done in the first pass to maintain the image quality.

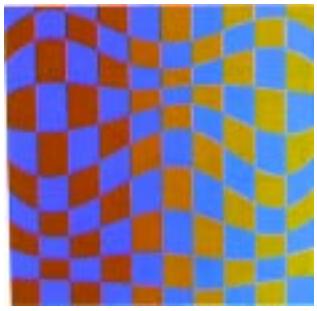


Figure 20a. In the two-pass system of Wolberg (1990), the image is first warped vertically.

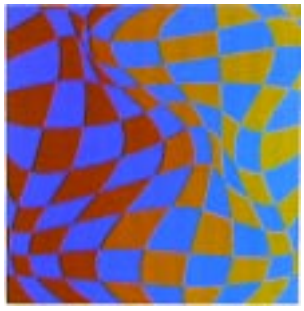


Figure 20b. In the second pass, the horizontal component of the warp is done.

Another system, a chip from Logic Devices Incorporated, has similar properties although it implements a second-order warp instead of a mesh warp. For our purposes these systems have a number of drawbacks. First, they cannot create the visibility gap that should be formed at depth discontinuities; instead they stretch that area in the image as if it were a single plane. Furthermore, they work with single flat images and therefore suffer from large visibility gap problems at the sides of the image (Figure 3). Finally, Wolbergs system is a two pass system that requires the first part to be finished before the second part can be initiated, which will result in long lags. Concluding, as such these systems are not useful for our purposes.

Another real-time system that can simulate camera rotations was build by Regan and Pose (1994). Their system is designed to handle surrounding images. They chose to map the surrounding image on a cube (see citation in the section 'Picture representation and consequences'). Figure 21a shows an unfolded cube, and Figure 21b shows an arbitrary view generated by their hardware.

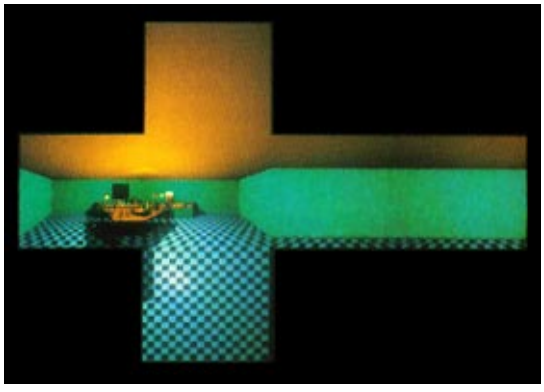


Figure 21a. Unfolded cube holding the surrounding image of a room.

Figure 21b. An arbitrary view generated with the hardware of Regan and Pose (1994) from Figure 21a.

Their system works by address recalculation (Figure 22). The pixels are calculated in scan line order. Given the pixel's screen coordinates, the direction of the light ray through that pixel is looked up in a lens lookup table, and corrected for the viewing direction of the observer. The memory location for that light direction is calculated. The display memory consists of 6 layers having a similar address, so

that each address results in 6 overlaying pixels. Finally, the non-transparent pixel that is closest to the observer is selected and displayed.

Each memory layer contains renderings of a fraction of the objects in the virtual world. Each object is put into a layer that fits best its rerendering requirements, for example fast moving objects should be rerendered more often than static objects far away. The first memory layer is rerendered with 60 Hz, the second with 30 Hz, etc. down to the sixth layer with 1.875 Hz. This is called 'priority rendering'.

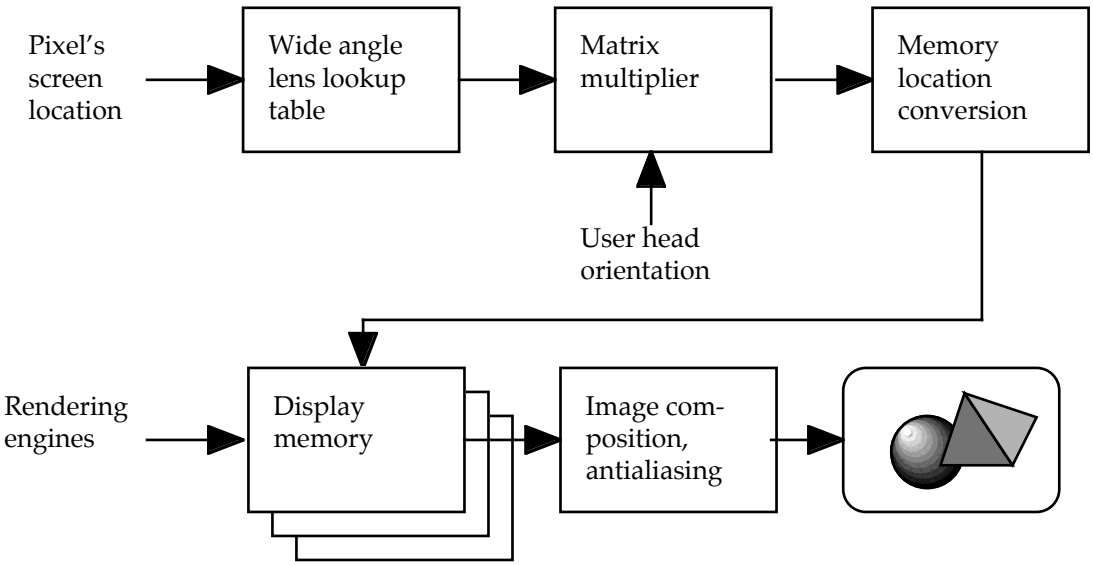


Figure 22. The address recalculation pipeline of Regan and Pose (1994).

If simulation of camera rotation alone would suffice for our project, this system would be a good choice, as it allows efficient use of polygon rendering in the headset. Close objects can be rerendered at a 60 Hz rate, giving minimal lags of at least 8 ms. This might be sufficient to simulate translation also, but it is dubious whether the minimal lag can be reached with a real polygon renderer. Another drawback is that a full view of the environment is stored for warping. This is essential for immersive VR, but is suboptimal for AR. Translation can be added in the warping system, but to avoid problems at the boundaries of each side of the cube the warp has to approximate the rules of perspective very close. Mark, McMillan and Bishop (1997) built such a system, and is discussed below.

Combined rotation and translation hardware

A system allowing both simulation of camera rotation and translation comes from Mann and Cohen-Or (1997). Their system is designed to work with a fast polygon renderer connected with a low bandwidth channel to the viewing system. The warping is being done on single images, as shown in Figure 12.

To resolve the resulting visibility gap problem (Figure 23a), Mann and Cohen-Or chose to extrapolate the foreground image instead of the background image (Figure 23b), resulting in objects that 'grow' into the visibility gap until correcting data is received.

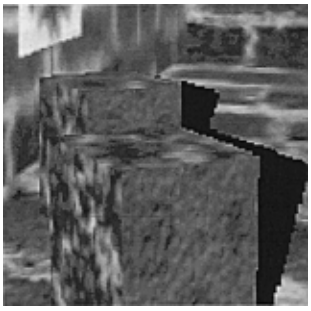


Figure 23a. See Figure 12 for the reference view. This figure again shows the warped image with the visibility gap.

Figure 23b. The visibility gap is filled with an extrapolated foreground image (from Mann and Cohen-Or, 1997).

In their system Mann and Cohen-Or regularly correct the accumulating warping errors, by doing the same warp on the polygon renderer and calculating the difference image with the fully rerendered image. This difference image is sent to the warping system to correct the warped image.

Judged from a video impression, the visibility gaps are quite disturbing. The eye is attracted to the growing and suddenly shrinking sides of objects. Furthermore, it is not clear what performance can be reached with this system. Finally, the system does much redundant work, as warps are performed in duplicate next to a complete polygon rendering. Concluding, the system seems less suited for image stabilisation in its present form.

A second system potentially capable of simulating both camera rotation and translation is the system of Mark, McMillan and Bishop (1997). Their system (Figure 24) generates warps two source images with partial depth information as in Figure 13b, by taking each image part only 1 pixel in size. Basically, the warpers in their system warp pixels based on their individual z-values, allowing for a warp that follows the rules of perspective projection. The two warped images are mixed in order to fill visibility gaps in one of the two images. The two source images are taken at the recent position and at an estimated future position of the observer. This way, the redundant information (as the two images usually will be nearly the same) is lowered provided that the observer actually reaches the predicted position.

This system suffers from 3 types of visibility gap problems. First, representing the scene as in Figure 13 implies in this case that visibility gaps may occur between each two pixels. Second, at real depth discontinuities image information may still be lacking. Third, the plain system warps flat images, and thus is prone to visibility gaps as shown in Figure 3.

To handle the first problem, Mark, McMillan and Bishop propose to use heuristics to find out whether adjacent pixels are from the same surface or not, They propose to compare the normal vectors at each two pixel's position. To handle the second problem, remaining visibility gaps are filled with an extrapolated background color. They claim that there will remain little visibility gaps, as long as observer stays on the line between the last reference image and the predicted future reference image.

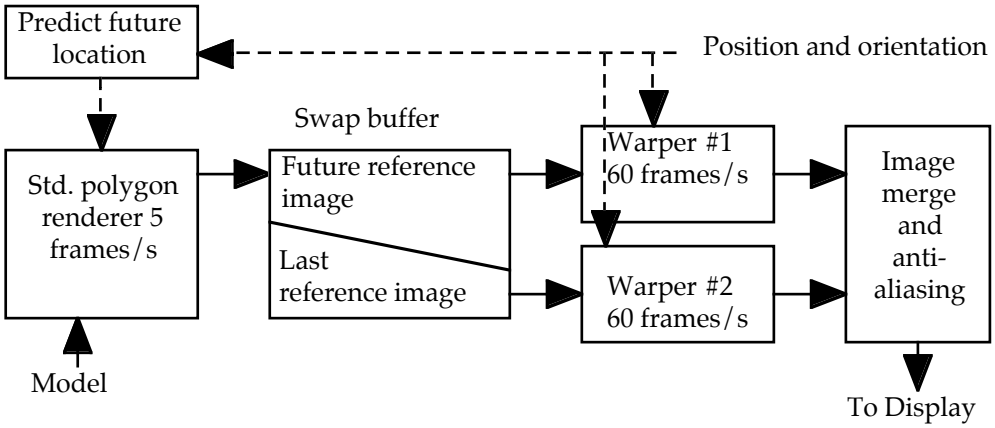


Figure 24. Conceptual diagram of the system of Mark, McMillan and Bishop (1997). For each layer on each side of the surrounding cube one such system is required. See text.

To handle the third problem, they suggest that the system can be used to warp four sides of a cube. It is not clear whether this option is standard for their system or only a proposal. But it suggests a combination of this warping system with a surrounding cube-system such as Regan and Pose (Figure 21). This is feasible, as the system of Mark, McMillan and Bishop gives warps that follow the rules of perspective exactly. Finally, it seems feasible to implement the system efficiently in hardware.

For our purposes, this system would allow for a nice upgrade path: start with the system of Regan and Pose and upgrade it with warping facilities in a later stage to allow for simulation of translation. Another nice feature is that this warp perfectly follows the rules of perspective, allowing exact alignment of virtual and real objects.

However, there are some drawbacks of this approach. First, double images with per-pixel z-data and per-pixel normal vectors are buffered. These images contain much redundant information. Second, two real-time warpers are required to generate a single image. Third, for a surrounding image six of these systems are required to warp each side of the cube, and additional hardware to build one final image from these six warped images: 6 sides x 2 positions x 6-fold storage per image (image + z-data + normal vector per pixel)=equivalent of 72 full-sized images. Combined, these three extra steps drastically increase the required hardware. Fourth, Mark, McMillan and Bishop are not clear about how the system would efficiently handle forward-backward translations (this seems to require a some devide instructions) and whether 6 full image buffers are required when the system is used to warp 6 sides of a cube (which might give a full frame delay). Finally, the system will not work when lags come above 200 ms (the prediction errors will be too high), which seems too conservative in our case. Concluding, this solution may work but is expensive.

Another system that allows both simulation of camera rotation and camera translation is the Talisman system (Lengyel and Snyder, 1997). It is not clear whether working prototype hardware exists for this system, but we can be sure that the hardware is complex as Samsung failed to build an essential piece of hardware (Vitaliano, 1997).

The principle of Talisman is that each object is rendered into a separate picture (*sprite*). The sprites can have transparent parts, so that each object can be fit into a

rectangular image. The hardware can warp each sprite independently with an affine warp in real time. The idea is that each object can be refreshed with an appropriate update rate, thus resulting in more efficient use of the polygon renderer. The sprites are then superimposed to form the image. The system claims to work in a 1024x768 pixels x24 bits resolution at 75 Hz, and requires a total bandwidth of 1.2 Gb/s to memory (Microsoft, 1996).

Figure 25 shows an overview of the Talisman hardware. The Polygon object processor renders separate objects from polygons into flat images (with a typical size of 128x128 pixels). Furthermore, these objects are not rendered to a single image but to chunks of 32x32 pixels. The resulting image chunks are compressed and put into the fast RAM. Thus, the polygon object processor is a traditional polygon renderer extended with compression hardware.

The image layer compositor reads the resulting images and performs real-time affine warping on each of them. Finally, the compositing buffer maps the resulting images for each object into a single output image. To reach the required memory bandwidth, all images (texture and rendered objects) are cut into 32x32 pixel chunks and compressed and decompressed in real-time (Microsoft, 1997).

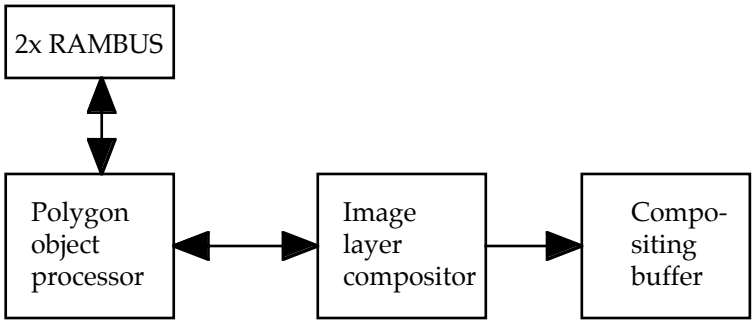


Figure 25. Overview of the Talisman hardware. See text.

The interesting part for us is the Image layer compositor. Torborg and Kajiya state that affine transformations can be done 10-20 times faster than polygon rendering. A buffer of 32 scanlines is required (giving a lag of 32/768 x (1/75) ≈0.5 ms). They are vague about the precise way the layering is implemented. Some address recalculation address mechanism might be used, but in that case smart caching mechanisms would be required to reach the required memory bandwidth. I estimate that a cache of 512K is required in order to reach the desired bandwidth.

A drawback of the Talisman system is that the field being warped essentially is flat. Therefore, the system can only partially simulate camera rotations, and it will suffer from visibility gap problems. Therefore, this design would need modification to be useful for image stabilisation.

A last system that can potentially simulate camera rotation and translation is the Quicktime VR system (Apple, 1998b). The Quicktime system as a whole is a collection of a large number of components that integrate audio, MPEG, normal and digital video, static pictures, animations, surrounding pictures, sprites, text, 3D objects, time codes etc., which may in itself be interesting for our project. However I quote the Quicktime system here for its surrounding-image capabilities (Apple, 1998a). This system uses a surrounding cylinder (Figure 4c).

Figure 26 shows an unfolded cylindrical representation of an environment. As indicated above, a cylinder does not cover the total environment; the bottom and top of the cylinder are omitted and rotations upwards and downwards are supported only partially. The latest version of the Quicktime system also supports sprites, and therefore might also be used to simulate camera translation. However, these sprites cannot be rotated, and therefore only partial rotation simulation is possible. As far as I know no special hardware exists to accelerate the display of surrounding Quicktime images, and software implementations reach refresh rates of about 5 frames per second (full-screen).

Figure 26. Surrounding image as used by the Quicktime VR system (Apple, 1998a).

Ideas for improvement

As they are, the discussed systems have drawbacks. The warping chip of Logic Devices Incorporated and the system of Wolberg can not handle surrounding images and thus are unable to simulate camera rotation. To simulate camera rotation the system of Regan and Pose (Figure 21) can be used, but simulation of camera translation is expensive in terms of processing power. The other systems allow only partial simulation of camera rotation, and simulation of rotation is essential for our purposes.

There are several ways in which the existing systems discussed above can be adapted to suit our purposes. I will discuss some of the possibilities below.

Adapting rotation-only hardware

A possibility is to upgrade the system of Regan and Pose. One way to add capabilities to simulate camera translation is the system of Mark, McMillan and Bishop (1997). But as discussed, this solution seems expensive. Another alternative may be to keep images of each object in a separate memory, and to copy these images into the warping memory when needed. This may be advantageous because this eliminates the need for 6 separate layers, saving hardware and power usage. But the problem with this warp stays the required precision of the warp to avoid glitches around the edges of the cube. Another disadvantage with this alternative solution is that it is not trivial to remove and replace an image once rendered into display memory. This problem is appearing similarly in graphical user interfaces such as X-windows (MIT?) and Finder (Apple, 1993).

Adapting Quicktime VR

Another solution is based on Quicktime VR with sprites. I assume that Quicktime VR uses sprites in polar form, but I did not yet find information about this. It is not clear whether useful warping exists for polar images (see above, Planar versus surrounding image). It seems appropriate to approximate the polar images with small rectangular images and use conventional warping.

This system may be implemented efficiently in hardware, but it will be of Talisman-like complexity. It requires a kind of address recalculation hardware and image composer. For each of the sprites, a warp can be calculated from the displacement of that sprite relative to the camera position. One approach is to calculate the required warp from the positions of certain points in the source images and the corresponding positions in 3D space. Another approach is to do only translation and scaling of the images to simulate camera translations, and to rotate the scene as a whole to simulate camera rotations.

A problem that needs to be solved in this system is the required bandwidth to memory. For each overlapping sprite the required bandwidth increases, and there may be too little time to wait for the front most sprite being fetched from memory and decide then whether a more distant sprite has to be fetched. As with Talisman, data prefetching may be required to prevent stalling of the rendering/warping pipeline. Instead of data prefetching, it seems possible and more straightforward to queue memory access instructions.

It may be a good idea to have fast access to low-resolution images of each sprite. Such low-resolution sprites may be used if the bandwidth to memory at some time is not sufficient to get a full resolution image. It may be also feasible to render the image at full resolution only at the place where the observer is looking at. Finally, when multiple sprites overlap, the low-resolution sprites may be used to estimate which sprites are really needed. With AR this there will be little overlap, probably only in 30% of the display given the factor of 1.7 for VR (Torborg & Kajiya, 1996), so this optimisation is not urgent.

A third way of reducing memory bandwidth is to compress the data before being put into memory, and to decompress it when needed. This approach also reduces the amount of memory required, and seems especially useful for image intensive operations such as VR. The feasibility depends on the availability of fast (hardware?) extraction of random pixels from a compressed image.

Figure 27 sketches a hardware layout that may do this warping. This layout calculates new warping parameters for each scan line. Probably this layout has to be extended with some caches. First, the warping parameters for each sprite are calculated given some characteristic points in the pictures and their corresponding spatial position, and the viewpoint and orientation of the observer. These parameters are fed to the machinery that calculates memory addresses containing the pixels needed for the warp. The address calculation may also perform lens distortion compensation. These source and destination addresses are written into a buffer. This buffer is needed because the sprite memory will take some time to read from. The sprite cache manager reads the source and destination addresses, checks whether they are in cache, and copies data to the pixel z-buffer. The pixel z-buffer selects the colour of the closest visible pixel. If data is not available it should be fetched from sprite image memory. Furthermore, the manager should manage the memory use in the cache. Of course the memory transfer manager should try to keep the sprite cache busy

given the data in the buffer, so that waiting for the main sprite memory is avoided. Once a line is completed, the line z buffer is swapped so that the next line can be transferred to the display. If the memory access times are extremely fluctuating more than two buffered lines may be required.

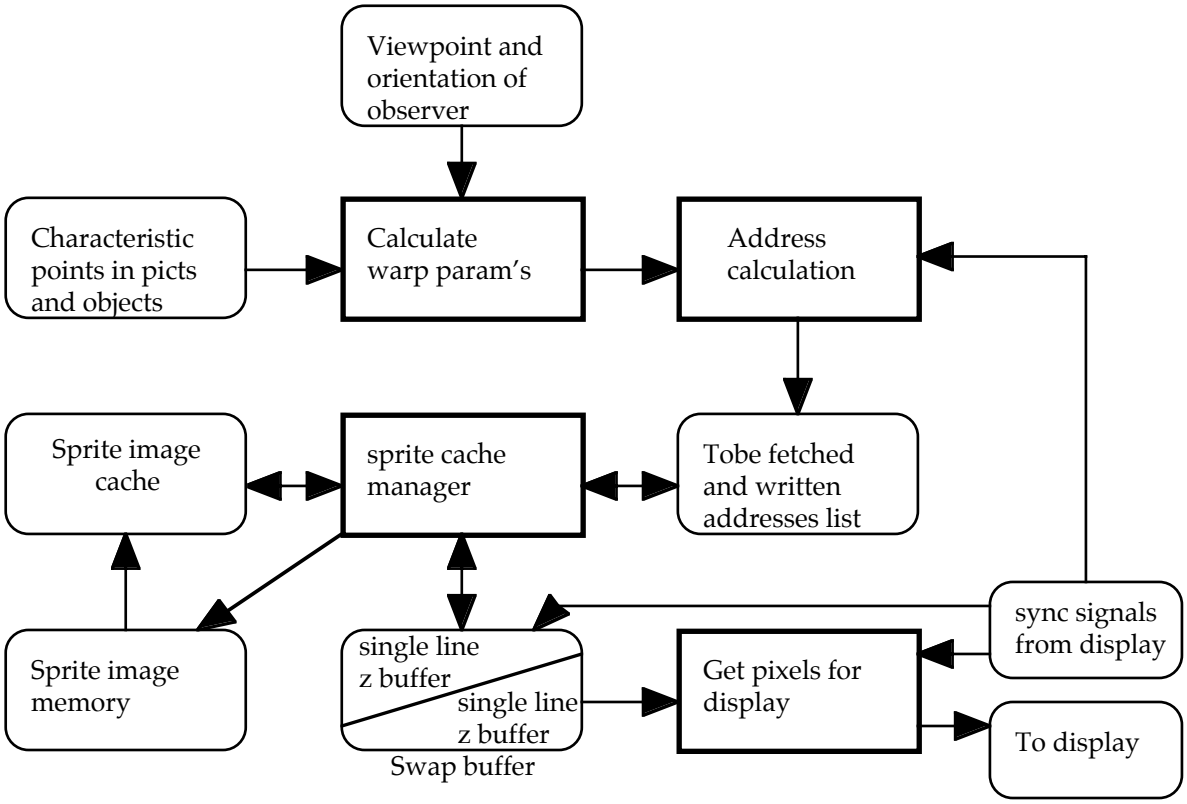


Figure 27. Possible warping set-up. See text. This set-up might be improved with compression, for example between the sprite image cache and the main sprite memory.

A more sophisticated set-up could, for example, use the low-resolution sprite data of the closest sprite to estimate which parts of more distant sprites is required.

Adapting Talisman

The Talisman system may be used for each of 6 planes of a surrounding cube to come to a surrounding view. However will cause irregularities at the edges of the planes, as Talisman only approximates a perspective projection. Using a polar representation will results in a design similar to the adapted Quicktime-VR above.

Brute force approach

It may be possible to take the approach of Figure 15 with standard rendering hardware, which requires brute force to render all the polygons with high refresh rates. With standard rendering hardware the display can usually be rendered in parts, for example the lower half of the display. When the number of overlapping parts to be rendered is not too high, this approach may be fast enough for our purposes. As shown in several games, such as Doom and Tomb Raider, the overlap in moderately complex scenes can be made very low, as a standard Intel-processor is fast enough to do full-screen polygon rendering of such scenes with a

refresh rate of about 10 Hz. From estimations done for the Talisman system (Torborg & Kajiya, 1996), an average overlap of 1.7 for realistic immersive VR scenes can be reached. As AR scenes are much less complex this approach may be realistic. However it will be hard to reach the required lag times of 10 ms, as Olano, Cohen, Mine and Bishop (1995) tried a similar approach and got lags of 17 ms with their 'Slats' system.

It may be advantageous here to have hardware to do the matrix multiplications and clipping too, as this may become a bottleneck in this scenario. It is unclear whether and to what extent clipping can be done in hardware, and to what extent the need for clipping can be avoided.

Overview

In the previous sections I discussed the possibilities to stabilise the image in an AR display. Table 1 gives an overview of the strong and weak aspects of the discussed systems. Note that the most promising systems, brute force polygon rendering and an adapted Quicktime VR system, do not yet exist.

Table 1. Overview of strong and weak aspects of discussed systems. +=system can potentially handle this; ±=system can handle this partially; -=system can not handle this; NR=not relevant.

	simul. of cam. rotation	simul. of cam. translation	visibility gap	lag <10ms possible	Working prototype hardware
Chip from Logic Devices	-	-	NR	+?	Yes
Surrounding Cube (Regan & Pose)	+	-	NR	+	Yes
Surrounding Cube (Mark, McMillan, Bishop)	+	+	±	+?	No
Quicktime VR	+	±	+	+	No
MPEG	-	±	-	±?	Yes
Wolberg's algorithm	-	±	-	-	No
Talisman	-	+	+	+	No
Brute force polygon rendering	+	+	+	±?	Yes
Adapted Quicktime VR	+	+	+	+	No

Proposal

As discussed, both camera rotations and camera translations have to be simulated in order to cope with the expected lags in the polygon rendering system. A system capable of simulating translations can simulate rotations partially, and the rotations it can not simulate may prove perceptually less urgent.

Another approach totally eliminating the need for real-time warping is the brute-force approach. What is needed to make this variant work is fast rendering hardware capable of rendering only part of the display in a fraction of the time it would take to render the entire display.

To estimate which technique is appropriate for our purposes, I propose to investigate the usefulness of three systems in more detail: the standard Quicktime VR with sprites, the adapted Quicktime VR approach with sprites warped with quadratic warping, and brute force polygon rendering.

To investigate the first two systems, I propose to make a video giving an visual impression of the performance of such systems. This video should superimpose warped images on real video images to give images comparable to what might be seen through an AR display. The video images should contain realistic movements and the warps should be based on these movements.

For the brute force system the question is not whether the quality of the images will be acceptable but rather whether a standard rendering system can be adapted in such a way that it fulfils our technical requirements. Low-level changes in the (complex) rendering pipelines is required to do this, and maybe highly system dependent. This approach will require extensive investigation of polygon rendering systems.

References

Apple (1993). *Inside Macintosh: Macintosh Toolbox Essentials*, Chapter 7 - Finder Interface. Addison-Wesley. Available Internet: http://devworld.apple.com/ngs/lpp/adrpub/docs/dev/techsupport/insidemac/Toolbox/Toolbox-443.html.

Apple (1998a). *Quicktime 3.0 preview: Programming with Quicktime VR 2.1. Panoramas*. Available Internet: http://devworld.apple.com/techinfo/techdocs/multimedia/qtdevdocs/VR/QTVRMgr.5.htm#pgfld=5564.

Apple (1998b). *Quicktime 3.0 Fact Sheet*. Available Internet: http://www.apple.com/quicktime/info/qt30/specsheet/qt3fact.html.

Azuma, R. and G. Bishop (1994). Improving static and dynamic registration in an optical see-through hmd. *Proceedings SIGGRAPH '94*, 197-204.

Deering, M. F., Schlapp, S. A., & Lavelle, M. G. (1994). FBRAM: A new form of memory optimized for 3D graphics. *Proceedings of the SIGGRAPH'94*, 167-174.

Fernando, G. M. X., Parker, D. W., & Rogers, P. T. (1990). Motion compensated display field rate conversion of bandwidth compressed HD-MAC Pictures. In L. Chiariglione (Ed.), *Signal Processing of HDTV, II*. Boston, MA: Elsevier Science Publishers B. V.

Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F. (2nd edition, 1987). *Computer graphics: Principles & Practice*. Reading, MA: Addison-Wesley.

Harrand, M., Mougeat, P., & Perron, C. (1992). A predictor IC for TV and HDTV codecs using motion compensation. In H. Yasuda & L. Chiariglione (Eds.), *Signal processing of HDTV, III*. Boston, MA: Elsevier Science Publishers B.V.

Hirose, M., Watanabe, S., & Endo, T. (1998). Generation of wide-range virtual spaces using photographic images. *Proceedings of the VRAIS* (March 14-18, Atlanta, Georgia), IEEE Computer Society, 234-241.

Hitachi (1998). *HB526C264EN-10IN, HB526C464EN-10IN: 1,048,576-word x 64-bit x 2-bank synchronous dynamic RAM module*. Available Internet: http://www.halsp.hitachi.com/tech_prod/m_memory/m_modules/6_x64/mm6td059d1/html/mm6h8.htm#32.
Jacoby, R. H., Adelstein, B. D., & Ellis, S. R. (1996). Improved temporal response in virtual environments through system hardware and software reorganization. *Proceedings of the SPIE* (28 January-2 February 1996, San Jose, CA), *2653*, 271-284. Partly available Internet: http://duchamp.arc.nasa.gov/research/latency.html.
Keran, C. M., Smith, T. J., Koehler, E. J., & Mathison, P. K. (1994). Behavioral control characteristics of performance under feedback delay. *Proceedings of the human factors and ergonomics society 38th annual meeting*, 1140-1144.
Lengyel, J., & Snyder, J. (1997). Rendering with coherent layers. *Proceedings of the SIGGRAPH'97*, 233-242.
Mann, Y., & Cohen-Or, D. (1997). Selective pixel transmission for navigating in remote virtual environments. *Eurographics'97*, *16* (3), C201-6.
Mark, W. R., McMillan, L., & Bishop, G. (1997). Post-rendering 3D warping. *Proceedings of the 1997 Symposium on Interactive 3D Graphics* (Providence, RI, April 27-30), 7-16. Available Internet: http://www.cs.unc.edu/~billmark/i3dwww/i3dpaper-web.pdf.
Microsoft, 1996.
Microsoft (1997). *Texture and rendering engine compression (TREC).* Available Internet: www.eu.microsoft.com/hwdev/devdes/WHNTREC.HTM.
Musgrave, F. K. (1992). A panoramic virtual screen for ray tracing. In D. Kirk (Ed.), *Graphic gems III*. Boston: Academis Press.
OKI Semiconductor (1998). *MSM5718B70 18-megabit RDRAM (2M x 9).* Available Internet: www.oki_europe.de/t-sync.htm.
Olano, M., Cohen, J., Mine, M., & Bishop, G. (1995). Combatting rendering latency. *Proceedings of the 1995 symposium on interactive 3D graphics* (Monterey, CA, April 9-12), 19-24 and 204. Available Internet: www.cs.unc.edu/volano/papers/latency.
Pasman, W., & Schaaf, A. van der (1998). *Prototype application.* Internal report, Delft University of Technology, Faculty of Information Systems and Technology, January.
Pasman, W. (1997). *Perceptual requirements and proposals for the UbiCom augmented reality display.* Internal report, Delft University of Technology, Faculty of Information Systems and Technology, December.
Regan, M., & Pose, R. (1994). Priority rendering with a virtual address recalculation pipeline. *Proceedings of the SIGGRAPH'94* (Orlando, FL, 24-29 July), 155-162.
Shade, Lischinski, Salesin, DeRose and Snyder, 1996
Torborg, J., & Kajiya, J. T. (1996). Talisman: Commodity realtime 3D graphics for the pc. Computer graphics proceedings. *Proceedings of the SIGGRAPH'96*, 353-363. Available Internet: www.research.microsoft.com/SIGGRAPH96/96/Talisman.
Vitaliano, F. (1998). *Intel MMX vs. Microsoft Talisman: Abbott and Costello do multimedia*. Available Internet: www.vxm.com/21R.98.html.
Wolberg, G. (1990). *Digital Image warping*. IEEE Computer society Press, Los Alamitos, CA.

Appendix A: What is warping

Warping is a two-dimensional transformation of an image into another image. The warps I will discuss in this appendix mainly follow Wolberg (1990). Conventionally, the rectangular source image is described with the parameters u and v, while the warped target image is described with the parameters x and y (Figure A1).

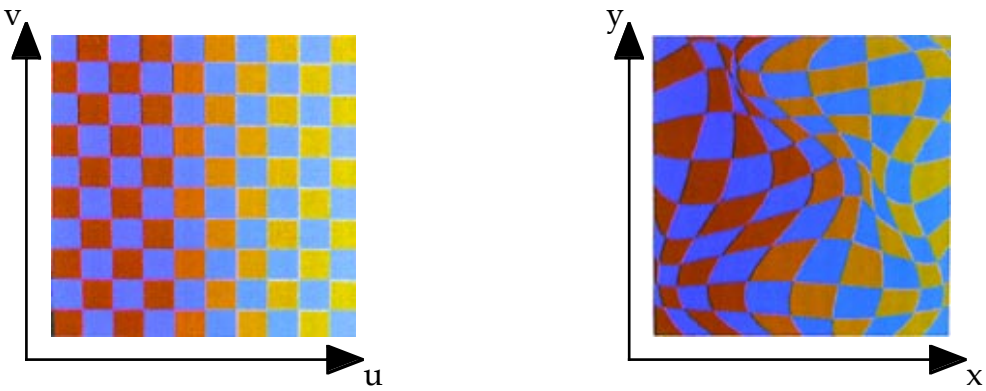


Figure A1. The square source image usually is described in (u,v) coordinate space, and the warped image in coordinate space (x,y). (from Wolberg, 1990).

Warping may involve just translation, rotation and scaling of a picture, but in more advanced kinds of warping parts of the picture may be stretched and others compressed. Some parts may be even hidden, which is called 'foldover' (see the second row of Figure A1).

Most types of warping are designed for planar images. However for AR purposes it seems more appropriate to use 'surrounding' images, such as the Quicktime VR images (Apple, 1998a). The possibilities are discussed under 'Planar versus surrounding image'. Furthermore, using a single image gives 'visibility gap' problems when approximating observer displacements. Some depth information of parts of the image seems needed, and is discussed.

When digital images are used, interpolation of pixels is required (Figure A2). This process is called 'resampling'. In this report I will not discuss resampling in detail, as it would complicate this discussion too much.

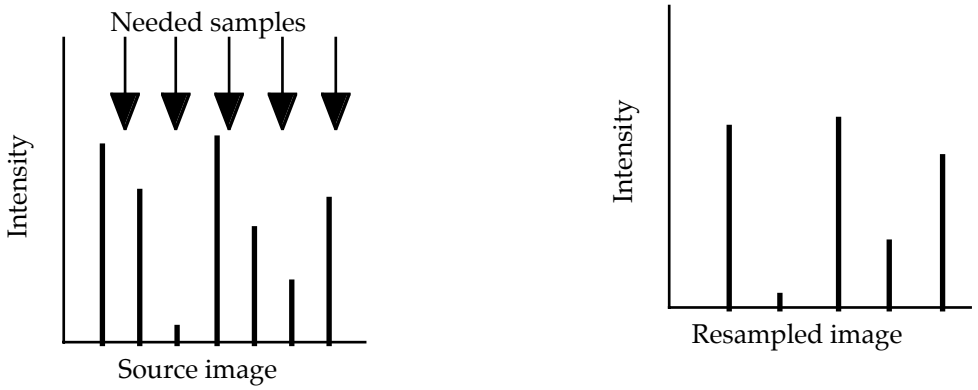


Figure A2. When the image is warped, the pixels needed in the warped image may come from positions between the pixels in the source image. Also, a pixel in the warped image may be a weighted average of several pixels in the source image. Such pixel calculation is called resampling.

Warping types

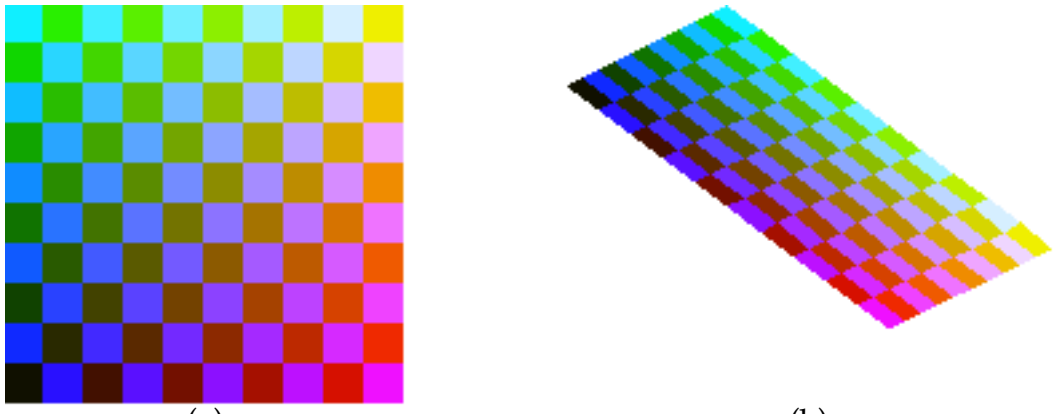
This section describes the main types of warping. Subsequent paragraphs will discuss their usefulness and feasibility for AR. Easy warps such as rotation, translation and scaling need no further explanation. The affine warp, the perspective warp and the mesh warp will be discussed.

Affine warp

With affine warps, the image point (u,v) is warped into the warped image (x,y) by the function

$$(x,y) = (Au + Bv + T_X, Cu + Dv + T_Y)$$

where A, B, C and D are constants and T_X and T_Y are translations. This allows for rotations, translations and shear (Figure A3).



(a) (b)

Figure A3. Demonstration of affine warps. (a) shows the image to be warped. Affine warps allow for rotation, translation, shear and combinations of these warps (b).

An affine warp is defined uniquely (and can be calculated in a straightforward way) if three points in an image (x1,y1), (x2,y2), (x3,y3) and their corresponding warped points (u1,v1), (u2,v2) and (u3,v3) are known.

Perspective warp

A perspective warp can be understood as an affine warp followed by a perspective transformation:

$$(x,y) = \frac{1}{Qu + Rv + S}\left(Au + Bv + T_X, Cu + Dv + T_Y\right)$$

As with the affine warp, A, B, C and D are constants and T_X and T_Y are translations. Q, R and S are the 'depth scaling' constants. This projection can be understood as follows. In a complete 3D perspective projection, a spatial point (x,y,z) is projected into image point (x/z,y/z) or written differently (x,y)/z. In the perspective warp, the z value is reconstructed by Qu+Rv+S, which is possible as the image is flat. Thus the perspective warp is the 2D equivalent of a perspective projection of a 3D square projected with the image on it. Figure A4 shows a typical perspective warp.

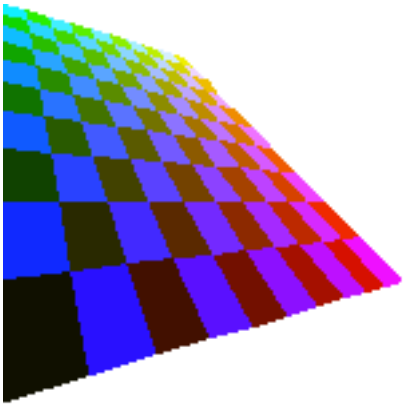


Figure A4. A perspective warp of a checkerboard as in Figure A3a is equivalent to a complete perspective projection of the image placed on a 3D square.

A perspective warp is uniquely determined when four points in the source image and their corresponding positions in the destination image are known. As with the affine warp, the parameters can be calculated in a straightforward way.

Both with perspective and affine warps, straight lines in the source image stay straight lines in the warped image. With more advanced warps, such as mesh warping, this is not the case. With mesh warping, parts of the image can be folded over other parts. In such cases, some priority has to be given for parts of the picture.

Bilinear and second-order transformation

The general form of a bilinear transformation (Wolberg, 1990) is of the form

$$x(u,v) = Bu + Cuv + Ev + F$$
$$y(u,v) = Hu + Kuv + Mv + N$$

where the parameters B, C, E, F, H, K, M and N are user defined. Figure A5. shows a typical bilinear warp. This form is popular because it avoids the divide operation required for perspective warps while still allowing to simulate

vanishing points. However, it can be seen that with bilinear warps equidistant points along the u-axis and v-axis remain equidistant in the warped image. For example, in Figure A5 the blocks in the top of the left row are as high as the blocks in the bottom of the row, while the higher blocks are further away and thus should be smaller.

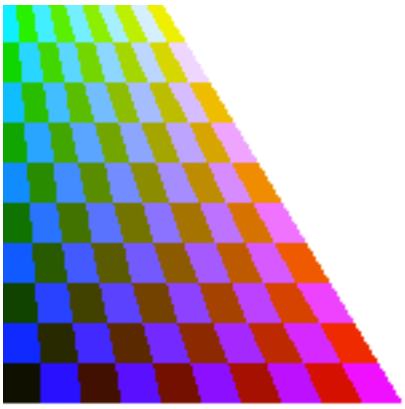


Figure A5. Typical bilinear warp.

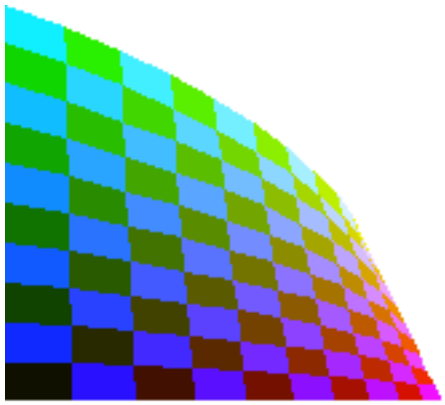


Figure A6. Typical second-order warp.

Introducing a u^2 and v^2 coefficient in the bilinear warp solves this equidistant-problem. Then we get the second-order warp:

$$x(u,v) = Au^2 + Bu + Cuv + Dv^2 + Ev + F$$
$$y(u,v) = Gu^2 + Hu + Kuv + Lv^2 + Mv + N$$

where parameters A through N of the transform are user-defined. With the second-order warp in the general case, lines will not be warped into lines (Figure A6). This can be seen from dy/dx (Equation 1). We can get straight lines by setting (G,H,K)=Q(A,B,C) where Q is some arbitrary constant (Figure A7). But we may not need straight lines as we only need an approximation of the full rendering.

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} = \frac{2Gu + H + Kv}{2Au + B + Cv} \qquad (1)$$

$$\frac{dy}{dx} = \frac{C_1 + Kv}{C_2 + Cv} \qquad (2)$$

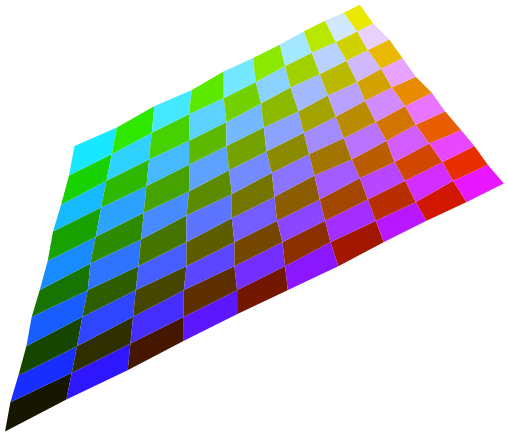


Figure A7. Second-order warp configured such that lines in the source image are warped into straight lines in the target image.

Mesh warping

With mesh warping, a number of control points can be placed in the source image. These control points may be a regular grid (Figure A8) but also a non-uniform mesh (Figure A9). Next, the warp is defined as a displacement of the control points.

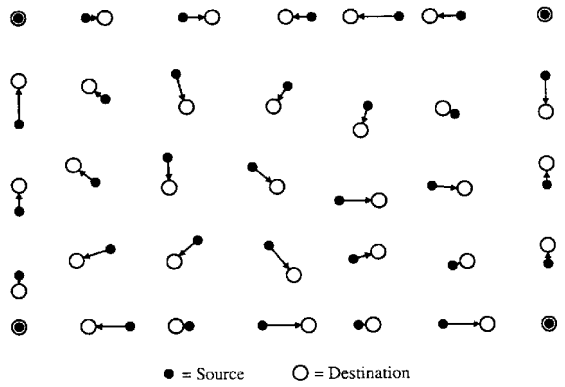


Figure A8. Control points in a regular grid.

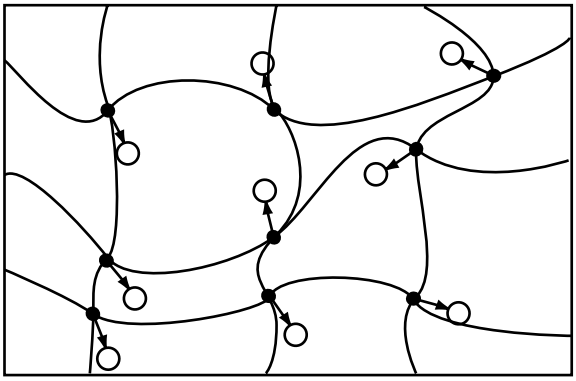


Figure A9. Control points in a non-uniform mesh.

Appendix B: cost estimations

This appendix gives rough estimations of the costs of warping and polygon rendering in terms of required processing power in the components and the bandwidth in their connections.

Figure B1 shows the sprite-based warping system that I proposed above in more detail. I assumed the following system properties:

- Display 640x400 pixels x 24 bits colour @ 100 Hz
- 50 sprites, average 50x50 pixels and 4 sprites of 200x200 pixels
- image storage ~260K in this scenario sufficient. 1 Mb seems reasonable.
- 5% of this (0.3 overlap) for AR, peaks of overlap 3 in some areas
- ~2 x 640 = 1300 pixels per line at most
- 100 x 400 = 40.000 lines per second = 52 Mpix/s

Furthermore I assumed that the divide operation costs 8 flops (Furthermore I think that flops are not appropriate to express differences in add, multiply and divide operations, but for now I will use them).

Note that the cache manager has a busy task. This may be the reason for Talisman to use prefetching instead of caching.

In case of a 4000x4000 display instead of 640x400 display many values are increased 60 times. This means peaks of 3 Gpix/s and averages of 300 MPix/s and a sprite memory of 16 Mb. At this moment this required bandwidth seems not feasible.

For the brute-force approach using a standard polygon renderer (Figure B2) I assumed similar system properties as for Figure B1:

- Display 640x400 pixels x 24 bits colour @ 100 Hz
- rendering the display in 3 blocks of 140 lines gives 3.33 ms lag
- 200 polygons and 300 vertices.
- 5% of this (0.3 overlap) for AR, peaks of overlap 3 in some areas

For both approaches, the main processing power and throughput is needed for the texture address generation and lookup. The main difference is in the way the addresses are looked up: in standard polygon rendering it is looked up per polygon, while it is looked up on a line-by-line basis in the warping approach.

In order to reduce the required texture pixels to be looked up and rendered, it may be possible (in both designs) to render only the part of the display the observer is currently looking at in full resolution, while displaying other parts of the display more at lower resolution.

Furthermore, in both designs cheap estimation of the warping/transformation parameters at the top level may be possible, reducing the amount of processing power at that point. However the savings will be marginal as compared to the texture address calculation power.

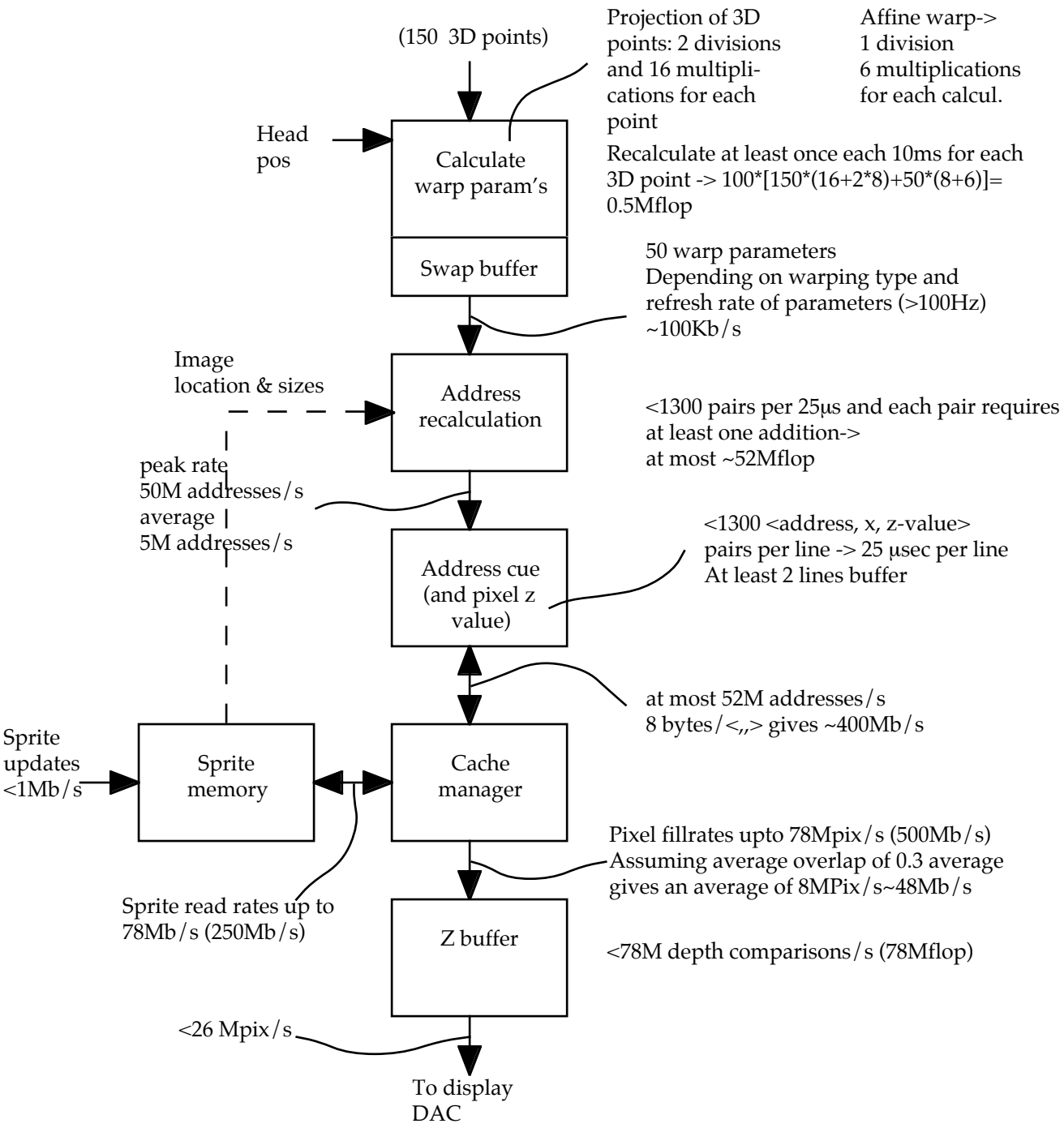


Figure B1. Cost estimation for the warping approach with surrounding sprites. See text.

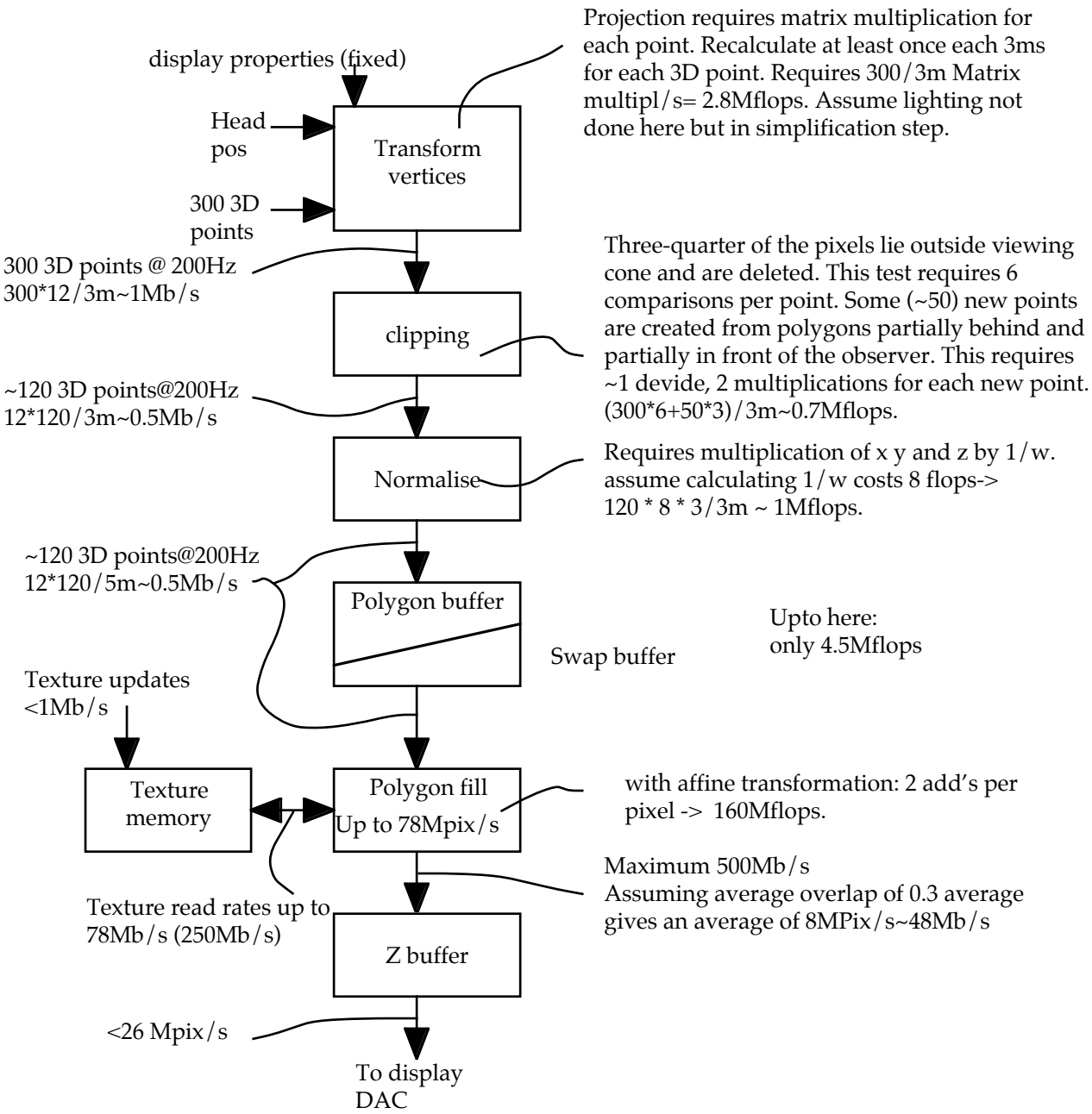


Figure B2. Cost estimation for brute force polygon rendering. See text.