Contents

1 Introduction **4**

2 Virtual Environments **6**

2.1 Introduction 6

2.2 Immersion 7

2.3 The Head-Mounted Display 8

3 Stereoscopy **11**

3.1 Background 11

3.2 Depth perception 13

3.3 Possible errors 14

3.3.1 General errors 15

3.3.2 HMD specific errors 19

3.4 A computational model for the optics in an HMD 22

3.4.1 Optics model for a single eye 22

3.4.2 Optics model for two eyes 24

4 The Stereoscopy Optimization System **33**

4.1 Introduction 33

4.2 Correcting the errors 33

4.2.1 General errors 34

4.2.2 HMD specific errors 34

4.3 Implementation 36

4.3.1 Hardware and software configuration 36

4.3.2 Initial calculations 37
4.3.3 Setting the output window 41
4.3.4 Setting up the transformations 42
4.3.5 Rendering the object 43
4.3.6 Precompute and predistort 46
4.3.7 Notes on extra features 51

5 Tests 54
5.1 Introduction 54
5.2 Depth in an HMD 54
5.3 Test description 56
5.3.1 Stereoscopic viewing test 56
5.3.2 IPD measurement 57
5.3.3 IPD test using special test object 57
5.3.4 Test using converging versus parallel viewlines 58
5.3.5 Predistortion test using regular grid 58
5.4 Test results 59

6 Discussion and conclusion 62

A Implementation aspects 65

B Figure sources 68

C Acknowledgements 69

Chapter 1

Introduction

In current Virtual Environment systems, the stereoscopic image generated using a Head-Mounted Display is far from optimal. Many parameters must be taken into account if the goal is to present a correct stereoscopic image to the viewer.

This report describes the issues involved when trying to achieve this goal. It is the result of a research project at the Physics and Electronics Laboratory of the Dutch organization for Applied Research, and also serves as a Master's Thesis for graduation at Leiden University.

The goals of our project were:

- identification of parameters influencing depth perception in a Head-Mounted Display
- development of a test system allowing independent manipulation of all parameters
- implementation of a test enabling setting of user-dependent parameters inside a Virtual Environment

The next chapter defines the typical Virtual Environment system setup and the role of the Head-Mounted Display. Chapter 3 describes the theory behind stereoscopic viewing. Section 3.2 illustrates the errors that occur if a parameter is not taken into account. Also a computational model is reviewed that incorporates the necessary parameters. In the following chapter the actual implementation of the test system is described. Chapter 5 considers the tests that were conducted with our system. A discussion and conclusion chapter ends the report.

The appendices contain the original test system requirements definition, the original design and specification, implementation aspects, the user manual, figure sources and acknowledgements.

Chapter 2

Virtual Environments

2.1 Introduction

Some people hold the opinion that our belief that we *are* somewhere, and that the things around us actually *exist*, is solely derived from our sensory perception. Our eyes see the surroundings, our ears hear the sound that is produced in it, our nerves signal that we are for example sitting in a chair, etcetera. This sensory information changes continuously in response to our actions, in a way we have grown accustomed to. If it is possible to replace our sensory data with synthetically generated data that responds to our actions the way we expect it to, we will perceive another environment. This is not a *real* environment, and is called a *virtual environment* (VE). The goal in a VE is to make us forget the environment we were in before we entered the virtual one. To summarize:

> A virtual environment is a simulation of an environment, created by influencing one or more human senses in the same way as in a real environment, in order to invoke a strong sense of being in another one.

2.2 Immersion

Of our five senses our visual system produces by far the most information, and consequently is the one that contributes most to our belief in reality. If we manage to "convince" our visual system, we will have succeeded in constructing the most important part of our simulation of reality. An important condition is that the perceived field of view (FOV) is large enough to make the viewer feel *immersed* in a virtual environment.

The size of the FOV our both eyes observe when looking straight ahead is about 180 degrees horizontally and 135 degrees vertically. A single eye has a horizontal FOV of about 150 degrees [Val66]. Approximately the central 20 degrees of each single eye FOV is projected onto the *fovea*, the most sensitive part of the *retina* (see Figure 2.1).[1] The visual axis is the line that connects the center of the eye lens and the center of the fovea.

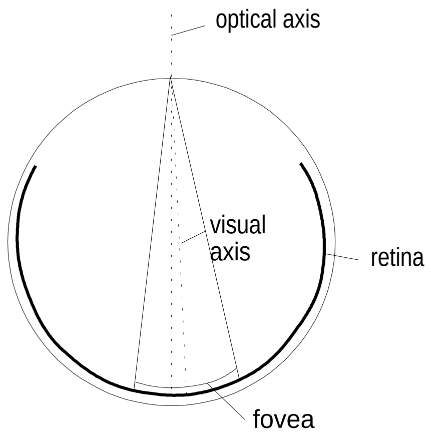

Figure 2.1: The retina and fovea in the eye

[1]The fovea is in fact a central area of 5.2 degrees, the next 3.4 degrees is the *parafovea* and the next 10.4 the *perifovea* [WS82].

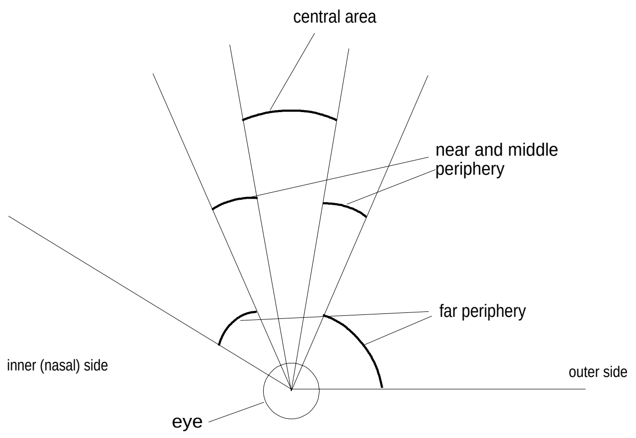

Figure 2.2: The various parts of the FOV

The *near periphery* subtends an angle of about 30 degrees, the *middle periphery* 50 degrees. The remaining 100 degrees are part of the *far periphery*, consisting of about 35 degrees on the inner (nasal) side of the eye, and 65 on the outer side (see Figure 2.2) [WS82]. Here viewing resolution is smallest, however it does serve in detecting movement.

Howlett suggests that in order to make a viewer feel immersed in a virtual environment, at least a binocular FOV of 90 degrees should be provided (see Figure 2.3) [How91]. [2]

2.3 The Head-Mounted Display

Currently the most popular tool used in generating an immersive visualization is a *Head-Mounted Display* (HMD). An HMD is a device that contains two small computer display screens, one for each eye. Between each eye and

[2]Compare the 10 degrees FOV of a televison (screen diagonal 67 cm, viewing distance 3 m).

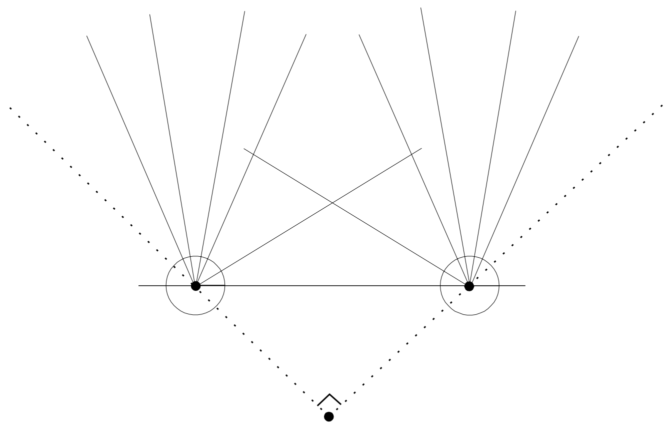

Figure 2.3: A binocular FOV of 90 degrees

its screen a lens system is positioned that magnifies the screen image, to enlarge the FOV. A position- and orientation-sensor is located on the HMD and continuously transmits its data to a computer. The computer incorporates this information of the viewer's position and viewing direction in the rendering process, and constructs a left- and right-eye image of a Virtual Environment. These images are in turn sent to the displays in the HMD (see Figure 2.4).

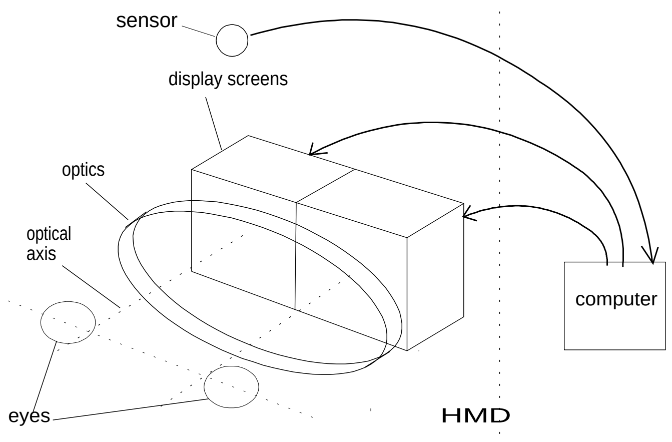

Figure 2.4: The HMD in a VE system

Chapter 3

Stereoscopy

3.1 Background

When human eyes look at an object in space, several factors contribute to the fact that they see a sharp, three-dimensional object [Hod92] [Fer87].

- The eyes turn towards the object: this is called *convergence of view-lines*, or *convergence* for short. The *convergence angle* is the angle between the viewlines.
- By bulging or flattening the eye lenses their focal length is changed, in order to focus on the object. This is called *accomodation.*
- The left- and right-eye are horizontally separated by the *Inter-Pupillary Distance* (IPD), hence see the object at a different angle. Consequently two (horizontally) different images are projected on the respective retinas. This is called *binocular disparity* or *retinal disparity.* The brain combines these images into one that is perceived as three-dimensional, or "having depth". This is called *stereo vision* or *stereopsis.* Depth is discussed further in Section 3.2.

If the images for the eyes are to be generated on a display screen, it is important to take the above factors into account. The result must be that

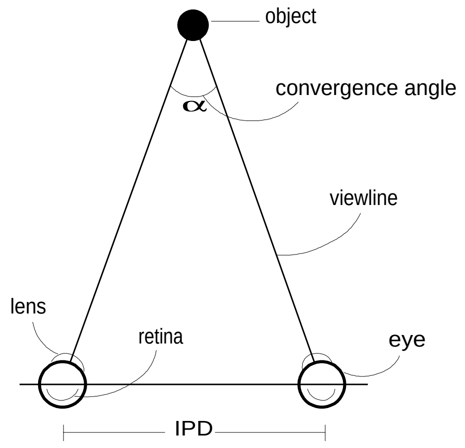

Figure 3.1: Eyes looking at an object in real life

orthostereoscopy is achieved. Robinett and Rolland define orthostereoscopy as

> constancy, as the head moves around, of the perceived size, shape and relative positions of the simulated objects [RR91].

Howlett formulates the same definition in terms of constancy of the azimuth and elevation of points [How91]. Sutherland recognized the importance of orthostereoscopy back in 1968:

> The image presented by the three-dimensional display must change in exactly the way that the image of a real object would change for similar motions of the user's head [Sut68].

In Section 3.3 a discussion is presented on the errors that can occur when attempting to achieve orthostereoscopy in an HMD. In Section 3.4 a computational model of an HMD is reviewed which has been used to implement orthostereoscopic image generation in our system.

3.2 Depth perception

As mentioned in the previous section, the image projected on the retina differs for each eye, the reason being that each eye looks at an object at a different angle.

When focus changes from a specific object to another object that is closer or further away, the convergence angle changes, resulting in a different horizontal disparity between the projected image in the left and right eye. If the difference is discernable by the eyes, the other object is *perceived* as being closer or further. The smallest observable difference (specified as the smallest convergence angle corresponding with the change that can be distinguished) is called the *stereo acuity*.

Assuming a certain stereo acuity α (say 1') and a certain IPD (say 65 mm), the largest distance D at which an object can be perceived as closer than an object at infinity can be calculated (see Figure 3.2). As can be seen $D \approx (IPD/\tan\alpha)$. For our mean values $D \approx 0.065/\tan 1' \approx 223$ m.

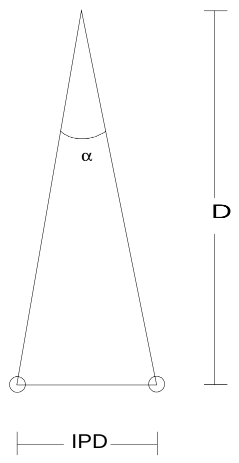

Figure 3.2: The furthest object in front of infinity

From an object at distance D we can move closer in angle steps of α, until we are at a distance of for instance 30 cm, and count each *depth step*: a depth step moves to the nearest point that can be perceived as being at another distance, hence the point differing α in convergence angle. Ferwerda uses the number of depth steps as a measure for the *amount of depth* in a stereo image: twice as many depth steps means twice as much depth [Fer87]. The number of depth steps s beyond a point at distance d can be calculated as follows:

$$\phi = \arctan(IPD/d)$$

$$s = \phi/\alpha$$

(with ϕ the convergence angle for distance d and α the stereo acuity). With a stereo acuity of 1', the number of depth steps beyond 30 cm is about 730. Note that the depth steps are smallest at close distances: about 92 % of all steps beyond 30 cm lie at distances closer than 4 meters. Consequently this is the most important area in depth perception.

In a stereo photo viewer, Ferwerda suggests a best obtainable acuity of 1.5', resulting in about 490 depth steps beyond 30 cm. So we can say that the amount of depth in a stereo photo is about two thirds of the depth in a real stereo image.

In section 5.2 we discuss depth in an HMD.

3.3 Possible errors

Currently in most VE systems stereo images are produced without taking many factors that influence the stereoscopic quality into account. Instead of measuring parts of the system to create a better image, the standard approach is that parameters are experimentally adjusted. This results in an image that has far from optimal depth. In this section we examine the errors that occur when using a standard approach.

The possible errors are classified into two categories:

- general, not HMD specific, errors
- HMD specific errors

Errors of the first category result from incorrect simulation of reality, in other words of the way eyes see in real life. Errors of the second category result from not or incorrectly incorporating the properties of the HMD itself in the rendering calculations.

3.3.1 General errors

accomodation does not correspond with convergence

Eyes are accustomed to convergence and accomodation being correlated: for every distance there is an appropriate convergence angle (such that the eyes are turned towards an object at that distance), and accomodation (to bring the object into focus)(see Figure 3.3).

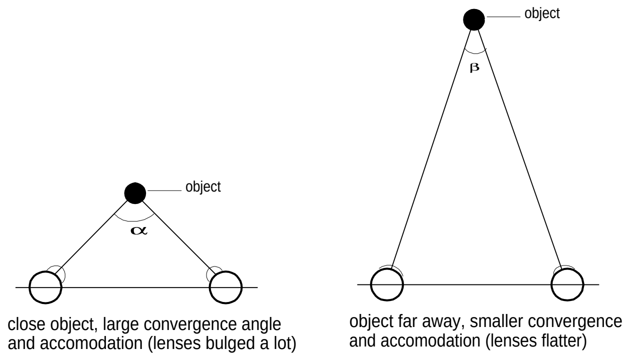
object
α
close object, large convergence angle and accomodation (lenses bulged a lot)
object
β
object far away, smaller convergence and accomodation (lenses flatter)

Figure 3.3: Convergence and corresponding accomodation

An image on a display screen normally is positioned at a fixed distance, hence the eyes have a constant accomodation. In Figure 3.4 the eyes converge to the image of the object, but must remain accomodated at the image of the screen.

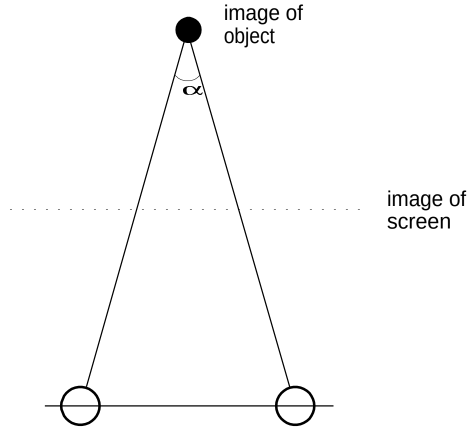

Figure 3.4: Constant accomodation on the image of the screen

Veron calls this phenomenon an *accomodation/convergence conflict* [VSLC90]. Valyus suggests a maximum allowable deviation from the proper convergence angle of 1.6 degrees.[1] Beyond that angle *doubling* may occur: although the intention is to make the viewer see for example one pixel, "the impression of two separate points is generated" [Val66]. Robinett and Rolland suggest the user must learn to decouple accomodation and convergence [RR91].

incorrect projection

- projection assuming parallel viewlines

  For effiency reasons, the perspective projection may be implemented assuming parallel view lines (see Figure 3.5). Note that this convergence angle *never* corresponds with reality, except when focus is at infinity (for instance, when looking at a star).

[1]This is a tight restriction: assuming an image distance of for instance 40 cm, the allowable object distance would be in the range 33 ... 48 cm.

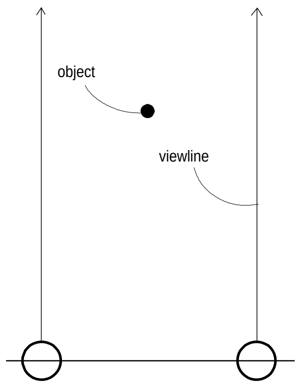

Figure 3.5: Parallel view lines

- on-axis projection

  The projection assuming parallel viewlines may be extended to account for the convergence error. Hodges describes an algorithm for this *on-axis projection* [Hod92]. He uses one center of projection and horizontal translations of the data. Roughly, the algorithm works as follows (see Figure 3.6):

  for the right-eye view:

  1. translate the object data to the left by $IPD/2$
  2. standard perspective projection
  3. pan the resulting image back

  and the other way round for the left-eye view.

  The Field Of View of on-axis projection is the same as for a single perspective projection. Williams and Parrish show that for example for a 40 degrees horizontal FOV per eye, the binocular FOV is 35 % smaller than it would be if an off-axis projection had been used [WP90].

- off-axis projection

  The *off-axis projection* most closely corresponds with reality, because it assumes *converging* instead of *parallel* view lines [Hod92]. The only problem here is to find out on which object the viewer is focusing, because this determines the convergence angle.

- rotation

  A method sometimes used to generate the left- and right-eye view is to rotate the left image counter-clockwise and the right image clockwise by a few degrees. This usually introduces vertical parallax (displacements) in the image, which causes severe eyestrain [Hod92].

incorrect Inter-Pupillary Distance

The IPD of a viewer determines how much he must converge his eyes to focus on an object at a specific distance. In Figure 3.7 this is illustrated by showing two viewers with a different IPD looking at an object at the same distance. If a standard IPD is assumed in the rendering computations (e.g. 65 mm), a viewer with a larger IPD would perceive the object at too large a distance, and someone with a smaller IPD would think the opposite.

3.3.2 HMD specific errors

Next the HMD specific errors are considered. Recall the VE system given in Chapter 2, shown again in Figure 3.8.

positional errors

If the optical axes were parallel, and passed through the center pixels of the screens and through the centers of the eyes, turning on the center pixels would show a dot positioned at infinity. But the axes may not be parallel, the screen centers may be offset from the axes, and the eyes may be offset as well.

- failure to account for angle between optical axes

  When the optical axes are not parallel, this has to be corrected by a rotation of the left- and right-eye image, such that it balances out the rotation. The situation is pictured in Figure 3.16 of Section 3.4.2, where a computational model for two eyes inside an HMD is reviewed.

- failure to incorporate position of screens

  If the screen centers are offset from the optical axes, all displayed data is offset. In case of a horizontal offset, the eyes need a different (incorrect) convergence angle to focus on an object. A vertical offset results in a height error.

- failure to incorporate Inter-Pupillary Distance

  In addition to using the IPD in the projection, it is also important with respect to the HMD. If the viewer has an IPD equal to the distance between the optical axes, the images are positioned correctly: each center of projection is located exactly in front of each eye. If the IPD differs from this distance, the images are in a horizontally incorrect position, resulting in a convergence error. In an HMD with mechanical IPD adjustment this problem does not occur, as the screens themselves are moved to get the centers positioned right.

- incorrect Field Of View

  The Field Of View used in the projection computations should be the same as the FOV actually experienced by the viewer, i.e. the FOV actually subtended by the images of the displays screens. If the computational FOV is too small, then the displayed object will appear too large, and vice versa.

There are two methods to determine the FOV. The first is to calculate it from the lens system specifications and the HMD build specifications. The second is analytical ray tracing through the optics: from the exact optics specifications (for each lens in the lens system) the exact path of a ray passing each lens surface is calculated. Robinett and Rolland used both methods to determine the FOV for the HMD and found a small difference [RR91]. The results of the second method can be found in their article. The first method is reviewed in Section 3.4.2.

optics errors

- non-linear distortion

  When a wide Field Of View is warped onto a flat plane, distortion is necessary [How91]. The LEEP optics used in many HMDs use a fish-eye like transformation to be able to project a large FOV onto a plane (see Figure 3.9). This means that the largest part of the image area is devoted to the central part of the FOV, and that the peripheral area is compressed into the side of the image. This corresponds with the relative importance of the various parts of the human FOV.

  When a flat plane is seen through the optics, the magnification is larger for points that are further from the optical axis. This is called a *positive* or *pin-cushion* distortion, and causes lines that are straight on the screen to be curved in the virtual image. Figure 3.10 is a graph showing the LEEP optics relative magnification for each distance from the optical axis (with the distance normalized between 0 and 1, 1 being the furthest point still viewable through the optics).

  In Section 3.4.1 a model for the distortion is discussed, as well as an approximate inverse distortion to correct the error (which is a *negative* or *barrel* distortion). The screen image is predistorted with this correctional distortion, in order to balance out the optics distortion. The effect of predistortion is shown in Figure 3.11.

- chromatic aberration

Differently coloured light rays diffract differently in the lens system, causing *lateral chromatism*, or "chromatic difference of magnification" [How91]. In the LEEP optics, blue is magnified about 1 % more than red, with green in between. This error is especially noticable in the peripheral part of the FOV.

3.4 A computational model for the optics in an HMD

In order to compute correct projections of the 3D image space, several HMD specific parameters need to be incorporated in the calculations. A computational model for the optics in an HMD given by Robinett and Rolland will aid in determining these parameters [RR91]. This model is extensively reviewed in this section.

3.4.1 Optics model for a single eye

First, an optics model for a single eye is given in Figure 3.12. From this model an approximation for the non-linear distortion is derived.

The variables shown in Figure 3.12 denote:

r_s radial position of a screen pixel, i.e. the distance between a pixel and the optical axis (point A_s in Figure 3.12)

r_v radial position of the virtual image of a screen pixel

In the specification of the optics the *maximum object field radius*, or w_s is given: this is the maximum distance between a point on the object and A_s, such that the point still can be seen through the optics. Its virtual image counterpart is w_v, the *maximum image field width*. Using these values, r_s and r_v are normalized:

$$r_{sn} = r_s/w_s$$

$$r_{vn} = r_v/w_v$$

If the optics had no distortion, then $r_{vn} = r_{sn}$. But they have, so a correction term must be added:

$$r_{vn} = r_{sn} + k_{vs} r_{sn}^3 \tag{3.1}$$

This is a 3rd-order polynomial approximation. The coefficient k_{vs} is a measure for the amount of distortion present.

Given that $r_{sn}^2 = x_{sn}^2 + y_{sn}^2$ and $r_{vn}^2 = x_{vn}^2 + y_{vn}^2$ (see Figure 3.13), Equation 3.1 can be written as:

$$(x_{vn}, y_{vn}) = \{(1 + k_{vs}(x_{sn}^2 + y_{sn}^2))x_{sn}, (1 + k_{vs}(x_{sn}^2 + y_{sn}^2))y_{sn}\} \tag{3.2}$$

These are the coordinates of a point in *image* space, expressed in its coordinates in *screen* space, taking distortion into account. However, we need to find the inverse of this function. An exact closed-form expression is not possible, but a 3rd-order polynomial approximation is:

$$r_{sn} = r_{vn} + k_{sv} r_{vn}^3 \tag{3.3}$$

Robinett and Rolland show that this approximation is at worst about 2 % off from the correct value[2]. Equation 3.3 can again be rewritten as:

$$(x_{sn}, y_{sn}) = \{(1 + k_{sv}(x_{vn}^2 + y_{vn}^2))x_{vn}, (1 + k_{sv}(x_{vn}^2 + y_{vn}^2))y_{vn}\} \tag{3.4}$$

The resulting screen coordinates are

$$(x_s, y_s) = (w_s x_{sn}, w_s y_{sn})$$

These have to be transformed into device coordinates using the offset of the screen center with respect to the optical axis, and the size of a pixel. There is a finite number of possible (x_s, y_s) values. Instead of computing these values every time a frame is generated, all corrected values can be precomputed and stored in a table. In Section 4.3.6 an implementation scheme for the precomputing and predistortion stages is discussed.

[2]This is an estimated error, measured from a graph in the article

3.4.2 Optics model for two eyes

The model discussed so far considered just one eye. We need to expand the model to include two eyes in order to

- calculate a correct FOV
- incorporate the offset of the screen centers with respect to the optical axes.

Calculate a correct FOV

From the positions of the display screens and the optics specification we can calculate the left- and right-FOV. Parameters that are used are:

T_y	vertical offset from the optical axis of top of display screen
B_y	idem bottom
I_x	horizontal offset from the optical axis of the inner side of the left display screen
O_x	idem outer side

The calculation is given below for T_y: [3].

1. $r_s = T_y$
2. $r_{sn} = r_s / w_s$
3. $r_{vn} = r_{sn} + k_{vs} r_{sn}^3$
4. $r_v = r_{vn} w_v$
5. $\phi = \arctan(r_v / z_v)$ (see Figure 3.14)

Given ϕ_T, ϕ_B, ϕ_I and ϕ_O we determine:

$$FOV_V = \phi_T + \phi_B$$

[3]Note that w_s, w_v and z_v depend on the *eye relief* d_{er}, the distance between the eye and the optics. The eye relief depends on the face shape of the viewer.

$$FOV_H = \phi_I + \phi_O$$

We also want to compute the binocular FOV (FOV_{bin}) and the overlapped FOV (FOV_{ov}). First we show the case with parallel optical axes in Figure 3.15.

Now we see that $FOV_{bin} = 2\phi_O$ and $FOV_{ov} = 2\phi_I$. When the optical axes are not parallel, these values are different (see Figure 3.16). Here $FOV_{bin} = 2\phi_O + \phi_{axes}$ and $FOV_{ov} = 2\phi_I \Leftrightarrow \phi_{axes}$. Looking at Figure 3.16 it is apparent that to correct for the rotated axes the right image must be rotated counter-clockwise, and the left image clockwise, both by $\phi/2$ through each eye.

Compare the binocular and overlapped FOV when the axes are rotated inward in Figure 3.17.

Incorporate the distance between the screen centers and the optical axes

Finally, a correction is needed if the center of a display screen is offset from the optical axis. To correct this error, a perspective projection is necessary that has its computational center of projection at that offset. Another method is to calculate the offset in pixels, and translate the screen image by that offset in the opposite direction. Note that the computed screen image must then be larger than the one shown on screen, or else data will be lost on the side of the image. Also there is a limit to the offset that can be corrected with this method.

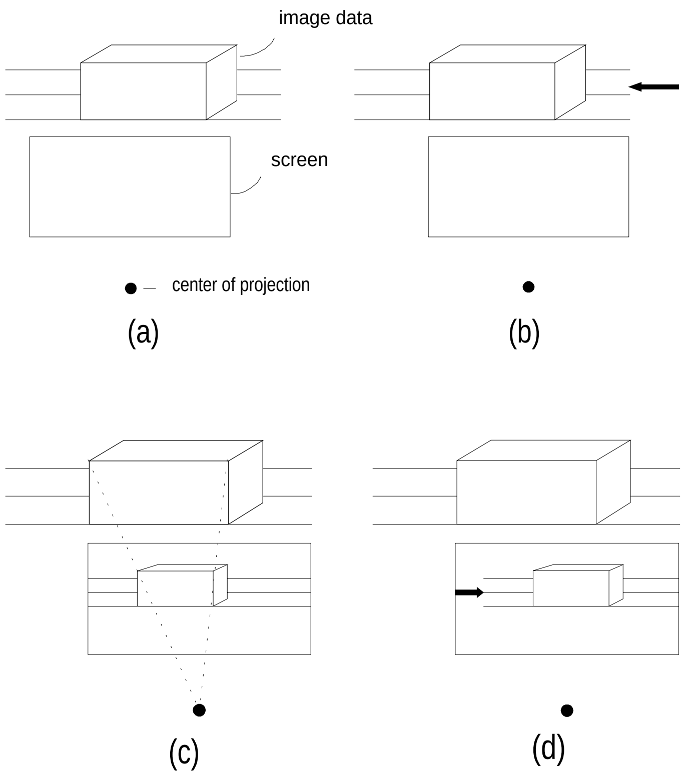

Figure 3.6: Initial situation (a), situation after translation (b), Perspective projection (c) and pan back (d)

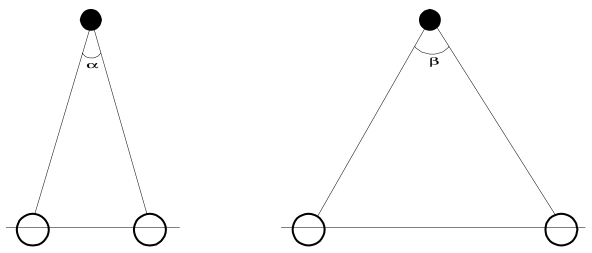

Figure 3.7: Different convergence because of different IPD

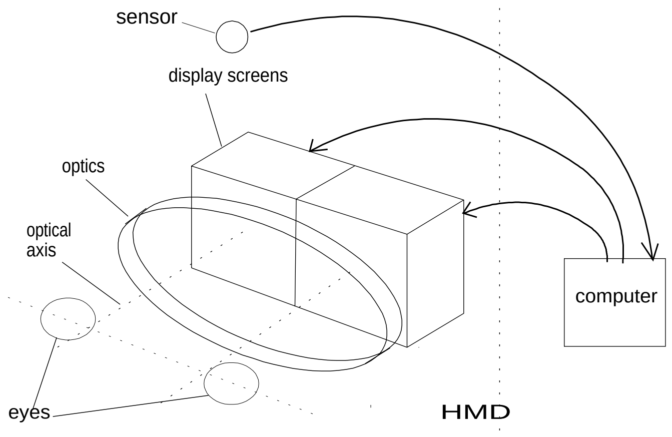

Figure 3.8: The HMD in a VE system

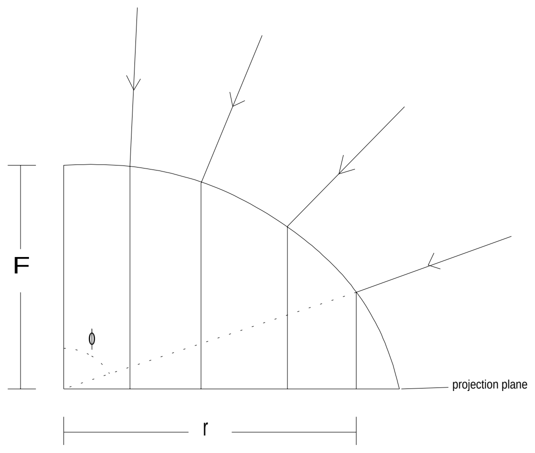

Figure 3.9: The LEEP optics projection

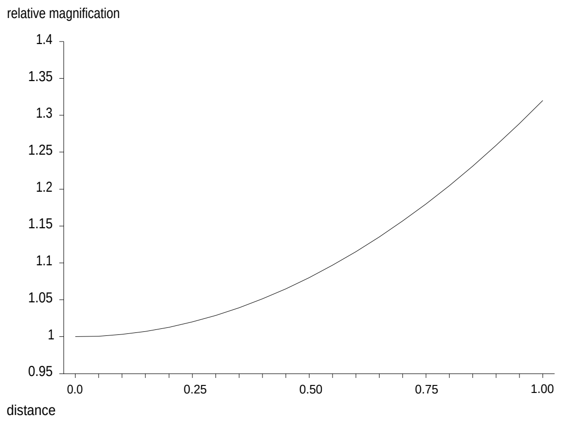

Figure 3.10: LEEP magnification in relation to distance from optical axis

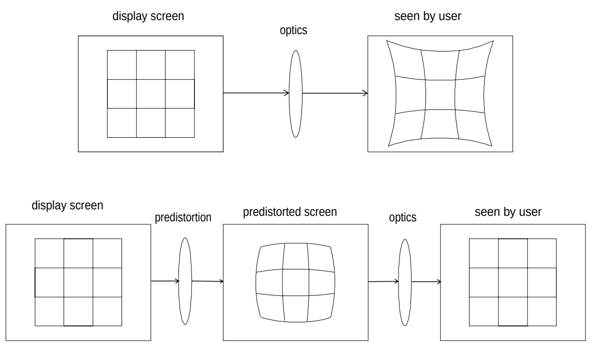

Figure 3.11: Results without and with predistortion

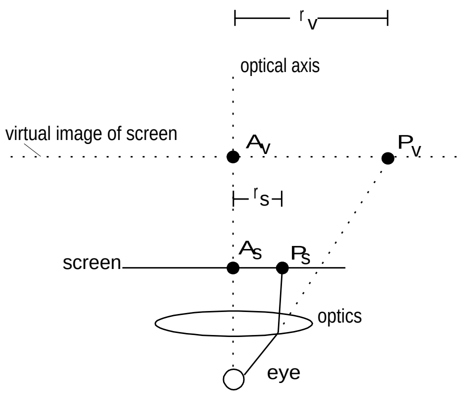

Figure 3.12: Optics model for a single eye

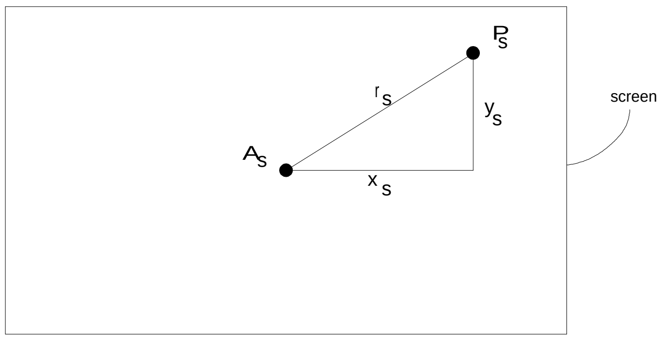

Figure 3.13: A point in screen space

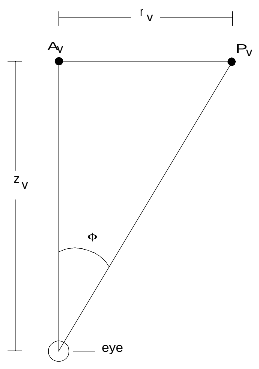

Figure 3.14: Angular position of a point

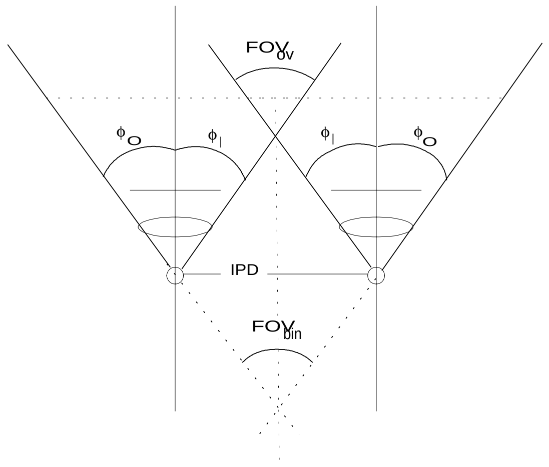

Figure 3.15: FOV with parallel optical axes

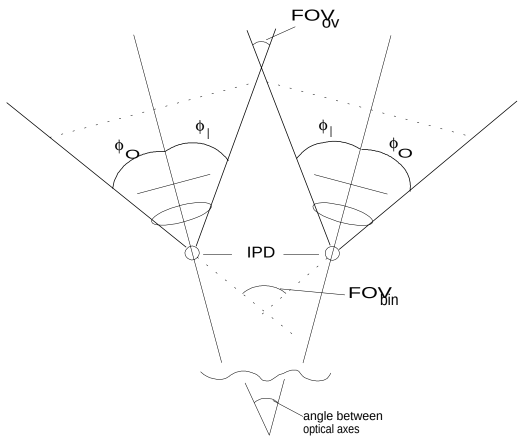

Figure 3.16: FOV with non-parallel optical axes, turned outward

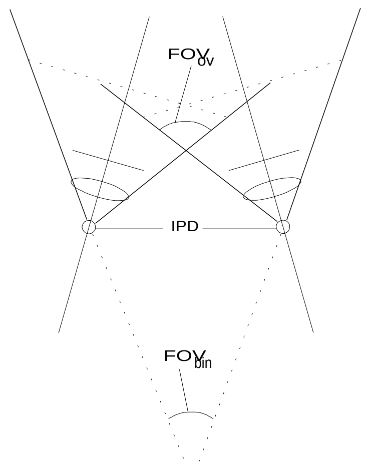

Figure 3.17: FOV with non-parallel optical axes, turned inward

Chapter 4

The Stereoscopy Optimization System

4.1 Introduction

To be able to determine the relative importance of the errors described in Section 3.3, a test system was developed that allows independent manipulation of many error-related parameters in a stereo image renderer. The system is called the *Stereoscopy Optimizaton System*, or SOS for short.

The error corrections that have been implemented in the SOS are described in Section 4.2. Section 4.3 examines the implementation of the corrections. More information on the implementation can be found in Appendix A.

4.2 Correcting the errors

This section discusses the error corrections that have been incorporated in our system, and the methods used to implement them.

4.2.1 General errors

accomodation does not correspond with convergence

This error cannot be corrected with our system, for two main reasons:

- First, the object on which the viewer is focusing must be known. This could perhaps be implemented using accurate eye-tracking devices.
- Second, this object would have to be rendered such that it is *in focus* if the viewer accomodates for the object's true distance, and all other objects in the environment *out of focus* (which is a simulation of *depth of field*), to a degree corresponding with their distance from the viewer. Due to limitations in display resolution and computing power, this cannot yet be done.

incorrect projection

The available projections in the SOS are on- or off-axis projection and projection assuming parallel viewlines. As there is always one object of interest in our environment, the focus is on this object when using off-axis projection. When using on- or off-axis projection, the resulting images are translated inward to cause the eyes to converge by the correct angle.

Attempting to generate a stereoscopic image by rotating the object has not been implemented, although for one object it is possible to generate a correct image (equivalent to the one using off-axis projection).

incorrect Inter-Pupillary Distance

The IPD can be changed, and is always incorporated in the rendering calculations.

4.2.2 HMD specific errors

positional errors

- failure to account for angle between optical axes

The correctional rotation (see also Figures 3.12 and 3.16) is always performed. The axes angle can be interactively changed.

- failure to incorporate position of screens

  To correct this error, the images are translated by such a distance that each computational center moves exactly on the optical axis.

  The images on the operator screen are larger than the ones seen inside the HMD. So when an image is translated to move the computational center at the optical axis, no image data will be lost (if the translation is not too large). The actual translation is performed by moving the origins of the images that are sent to the HMD. The computational FOV is scaled up to account for the larger operator screen images.

  Note that we must know exactly which part of each image on the operator screen appears on the screens in the HMD, in order to determine the translation in operator screen pixels.

- failure to incorporate eye positions

  If the IPD is different from the distance between the optical axes, a horizontal translation is performed to move the computational center of projection exactly in front of the eyes.

incorrect Field Of View

The FOV is calculated using the formulas described in Section 3.4.2. The necessary parameters are read from an *HMD file*, which contains all HMD specific data. As with the IPD, the vertical and/or horizontal FOV can be changed while the images are continously updated.

optics errors

- non-linear distortion

  The predistortion formula of Section 3.4 has been used. As has been said, there is a finite number of possible (x, y) screen coordinates, hence their predistorted values are precomputed and stored in a table.

- chromatic aberration

  Correction of the chromatic aberration is not implemented. A way to do this could be to render the red, green and blue components of the object in three separate frame buffers, scaled by the correct amount to compensate for the aberration, and then combine them.

4.3 Implementation

4.3.1 Hardware and software configuration

The SOS is implemented using the following hardware:

- Silicon Graphics 4D-240VGX graphics workstation with
  - videosplitter
  - four 25 Mhz MIPS 3000 RISC processors
  - 48 MBytes of main memory
- Virtual Research Flight Helmet

The videosplitter produces output of four arbitrary 640x485 sections of the 1280x1024 operator screen. Two of these are sent to the HMD. [1]

The development software used:

- operating system: Irix version 4.0.5
- compiler: Gnu C++ version 2.3.3
- graphics library: GL
- user interface library: Forms Library version 2.1 [Ove92]

[1]The Videosplitter output is RGB with NTSC (RS170A) timing. Two outputs are connected to a converter that produces two NTSC composite signals which are sent to the HMD.

4.3.2 Initial calculations

In class *object*

The point with average x, y and z coordinates and the point with maximum coordinates are calculated. The average point will become the point on which the viewer focuses.

In class *viewer*

The mentioned object points are used to set an initial viewer position. The eyes are offset relative to this position, parallel to the XOZ plane (note that the y-axis is pointing upward). The azimuth and elevation of the viewer are computed, which are used when the viewer needs to be rotated.

In class *HMD*

All HMD specific information needed in the calculations is stored in an *HMD file*. This is a plain text file. The file format is illustrated by the example given below.

```
flhelmet.hmd -


/*

   comments:

   HMD file for Virtual Research Flight Helmet

*/


{           /* begin file with a { */


/*

   all variables are on a separate line, of the form:

   variable = value

   all distances are in meters, case is not significant

*/
```

```
-

/*
   true Field Of View, as may be determined by analytical ray tracing
   through the lens system.
   These variables may be left out if unknown,
   all other variables _must_ be present
*/
TrueVerticalFOV = 58.4
TrueHorizontalFOV = 75.3
```

```
-

Ty = 0.0218        /* offset of top of screen from optical axis */

By = -0.0185       /* idem bottom of screen */

Ix = 0.0209        /* idem inner side of left screen */

Ox = -0.0333       /* idem outer side of left screen */

Cx = -0.0062       /* horizontal center offset from optical axis

                      of left screen */

Cy = 0.00165       /* vertical center offset */
```

```
-

Magnification = 9.66  /* transversal magnification of the optics */

VirtualImageDistance = 0.3982   /* distance of virtual image     */

ObjectFieldRadius = 0.0281      /* maximum object field radius   */

OpticalAxesAngle = 0.0          /* angle between optical axes    */

OpticalAxesDistance = 0.064     /* distance between optical axes */

DistortionCoefficient = 0.32    /* coefficient of optical distortion    */

InverseDistortionCoefficient = -0.18  /* inverse distortion coefficient */


}   /* end file with a } */
```

First the various FOVs are calculated using the formulas described in Section 3.4 and the values read in the HMD file. Next the horizontal and vertical screen center offset read in the HMD file are converted to offsets in operator screen pixels. Finally the predistortion tables are precomputed, which is discussed along with the predistortion in Section 4.3.6.

4.3.3 Setting the output window

This operation corresponds to the data transformation *set image origins* in Figure **??**, and is used whenever one of the following variables is changed:

- IPD
- axes distance
- screen center offset
- distance from viewer to object (when using on- or off-axis projection)

Our rendering computer has a hardware card called *videosplitter* that enables independent output of four quadrants of the operator screen. Two of these outputs are used to send images to the HMD. The initial origins are set such that the regions sent to the HMD are centered in the operator screen images. When all of the above parameters are incorporated, the origins of the regions are set such that the computational center of projection is in front of each eye. The correction of an offset screen center is called an *off center correction.* [2]

4.3.4 Setting up the transformations

Projection transformation

Now we are ready to set up a correct projection and viewing transformation. First the perspective projection is determined. As only part of the image windows on screen are sent to the HMD, the FOV which is specified to the `perspective` (a GL function) projection must be adjusted to account for the size difference. This is done using the equations given below.

- $width_{VS}$ is the width in pixels of the part of the screen image sent to the Videosplitter (which in turn sends it to the HMD)
- $width_{image}$ the width in pixels of the total screen image,

[2]Note that we do not correct the vertical screen center offset: the vertical error than can be corrected is small because the operator screen images are just 27 pixels heigher than the images sent to the HMD. The vertical error is much less important than the horizontal one, as it results in only a height error.

- FOV_{hor} is the current monocular horizontal FOV as perceived in the HMD
- FOV_{adj} is the corrected FOV, such that the computational FOV in the part of the image sent to the HMD equals FOV_{hor}

$$\left.\begin{array}{l}\tan(FOV_{hor}/2) = width_{VS}/2z_v \\ \tan(FOV_{adj}/2) = width_{image}/2z_v\end{array}\right\} \Rightarrow$$

$$\tan(FOV_{adj}/2)/\tan(FOV_{hor}/2) = width_{image}/width_{VS} \Leftrightarrow$$

$$FOV_{adj} = 2\arctan(\tan(FOV_{hor}/2)(width_{image}/width_{VS}))$$

Viewing transformation

The viewing transformation is set using the GL function `lookat`, which takes as parameters the eye position and the point the eye is looking at. Hence converging or parallel viewlines are easily implemented.

Axes angle correction

Directly after `lookat()` is called, a rotation matrix is added to correct for a possible angle between the optical axes. Recall from Section 3.4.2 that to correct for the rotated axes the right image must be rotated counter-clockwise, and the left image clockwise, both by half the axes angle through each eye.

4.3.5 Rendering the object

The object file format

The object file format being supported is the *PAZ* format, as used by Division in their VE systems. The formats supports the *tristrip* and *polystrip*, which are both triangular meshes, as in Figure 4.1.

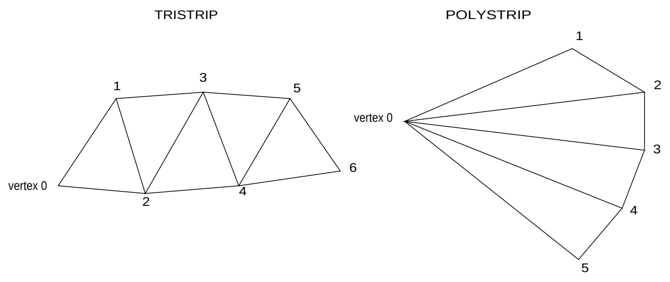

Figure 4.1: An example tristrip and polystrip

Vertices 0, 1 and 2 are the same for both strips, but for $n > 2$ triangle number n is built from vertices $n \Leftrightarrow 2$, $n \Leftrightarrow 1$ and n for a tristrip, and vertices $n \Leftrightarrow 2$, $n \Leftrightarrow 1$ and 0 for a polystrip [Atk92].

One object consists of one or more patches, and one patch of one or more strips. For the object and/or for specific patches several surface variables may be set. They are listed in Table 4.1 [3].

In the SOS all variables are parsed and stored, but only variables COLOUR, COOKED and NORMALS are supported.

An informal description of the PAZ file format is given in the dVS system documentation [Atk92]. In Appendix A a formal LL(1)-grammar of the PAZ file format in BNF is given, which was used to facilitate the implementation of a structured and easily extensible parser [Ter86].

[3]The table is copied from [Atk92]

variable	default value	meaning
COLOUR	{1.0, 1.0, 1.0}	diffuse colour of front surface, RGB triplet
B_COLOUR	{0.0, 0.0, 0.0}	diffuse colour of back surface
KS	0.0	specular reflection coeff. of front surface, 0.0 ...1.0
B_KS	0.0	specular reflection coeff. of back surface
POWER	0.0	specular lighting exponent of front surface, 0.0 ...32.0
B_POWER	0.0	specular lighting exponent of back surface
NORMALS	0	the patch has vertex normals embedded in the structure
COOKED	0	the patch has colours embedded in the structure
TEXTURE	0	the patch has texture u,v coordinates embedded

Table 4.1: PAZ variables

The parser is implemented in class `datafile`: it tokenizes a text input into identifiers, numbers or delimiters. Besides the `object` class, class `viewer` uses `datafile` for saving and loading of the file holding a viewer's IPD. Class `hmd` uses it for the HMD file described previously.

The object data

The Graphics Library (GL) available on our machine supports direct rendering of the trimesh and polymesh. A trimesh is rendered by calling `begintmesh()`, sending all vertices in order, and calling `endtmesh()`. A polymesh is in fact a trimesh having the last two vertices swapped each time a vertex is added. The call `swaptmesh()` uses this fact.

So the object data structure can be implemented using one linked list, each element holding a vertex or a delimiter between strips.

The actual object rendering (i.e. the sending of the object data to the graphics hardware) is done by a traversal of the vertices list.

4.3.6 Precompute and predistort

Precomputing the predistortion tables

As has been said in Section 3.4.1, there is of course a finite number of (x_s, y_s) values (i.e. pixels) that have to be moved to another location. Hence all destination coordinates can be precomputed and stored in a table. Note that we need a table for both the left and right image, as the optical axis is in a different position in each screen [4]. In our configuration, with an image resolution of 640x485, the tables each use just under 1.2 Mbytes of memory.

First the position of the left and right optical axes (measured from the bottom left of each screen) is determined. Next a loop is entered in which for each (x_s, y_s) couple of the left and the right image its destination coordinates are calculated, which are stored in their respective tables. The source listing below only shows the calculations for the left image values: obviously in the actual program the right image values are computed as well.

[4]It is also possible to use one table, which is large enough to hold every possible offset from an optical axis.

hmdcalc.cc - precompute()

```
for(int y_d = 0; y_d < ver_res; y_d++)

{

  for(int x_d = 0; x_d < hor_res; x_d++)

  {



// compute offset from optical axis



    float left_x_s  = x_d * pixel_width  - left_axis_x;

    float left_y_s  = y_d * pixel_height - left_axis_y;



// normalize these values



    float left_x_sn  = (left_x_s  / w_s);

    float left_y_sn  = (left_y_s  / w_s);
```

```
-

// square the radial distance

    float left_r_sn_sq  = left_x_sn  * left_x_sn  + left_y_sn  * left_y_sn;

// compute new normalized value using inverse distortion coefficient k_sv

    float left_x_sn_new  = left_x_sn  + k_sv * left_x_sn  * left_r_sn_sq;
    float left_y_sn_new  = left_y_sn  + k_sv * left_y_sn  * left_r_sn_sq;

// compute new radial distance

    float left_x_s_new  = left_x_sn_new  * w_s;
    float left_y_s_new  = left_y_sn_new  * w_s;
```

```
-

// compute new pixel coordinates

    float left_x_d  = (left_x_s_new  + left_axis_x)  / pixel_width;
    float left_y_d  = (left_y_s_new  + left_axis_y)  / pixel_height;

// check if anything moved out of the window (which will happen if we use
// a pin-cushion distortion to correct for a barrel distortion in an HMD)

    if (left_x_d < 0) left_x_d = 0;
    else if (left_x_d > hor_res) left_x_d = hor_res;

    if (left_y_d < 0) left_y_d = 0;
    else if (left_y_d > ver_res) left_y_d = ver_res;

// store the resulting values in their tables

    long index = ((y_d * hor_res) + x_d) * 2;
    left_predist_table [index] = (short) left_x_d;
    left_predist_table [index + 1] = (short) left_y_d;
  }  // for x_d
}  // for y_d
```

Predistortion

Now we have two tables containing destination coordinates for each pixel in our left and right images. The predistortion takes place after the entire rendering loop, just before both images are made visible. [5] It is implemented in a separate `predist` class, which is a data member of the `hmd` class. It holds arrays for the old (not predistorted) and the new (predistorted) left and right image, and the old and new left and right z-buffers. These arrays each take also just under 1.2 MBytes of memory. So the total memory requirements for this predistortion implementation (including the precomputed tables) are a little under 12 MBytes.

The actual predistortion algorithm is fairly straigt-forward:

- both images and z-buffers are read from the back frame buffer
- Next the destination arrays and pointers to the predistortion tables are initialized (i.e. the destination images are zeroed and the destination z-buffers are set to maximum value)
- Then the predistortion loop is entered. For each left and right image pixel its new coordinates are fetched from the predistortion tables. The pixel is then copied to its new destination if its z-buffer value is less than the current value. This is necessary when the predistortion is a barrel distortion: then usually more source pixels map onto one destination pixel. [6]
- the resulting image arrays are copied back to the frame buffer, which is then made visible

[5]The images are rendered into the *back frame buffer*, which is not visible on screen. Each frame is made visible by swapping the back and the *front* frame buffer.

[6]If the predistortion coefficient is made positive, i.e. the predistortion is a pin-cushion distortion, loss of data will occur because data is mapped onto a larger area. The visual result (gaps in the image) may be alleviated using an appropriate filter.

4.3.7 Notes on extra features

Movemode

In this mode changes in mouse movement are translated to viewer movement, until the middle mouse button is pressed. After a new object has been loaded, the initial viewer position is set to be some distance from the object's center, proportional to the object's diameter. The eyes are positioned on a line perpendicular to the y-axis and the viewline (see Figure 4.2). Note that the y-axis is pointing upward.

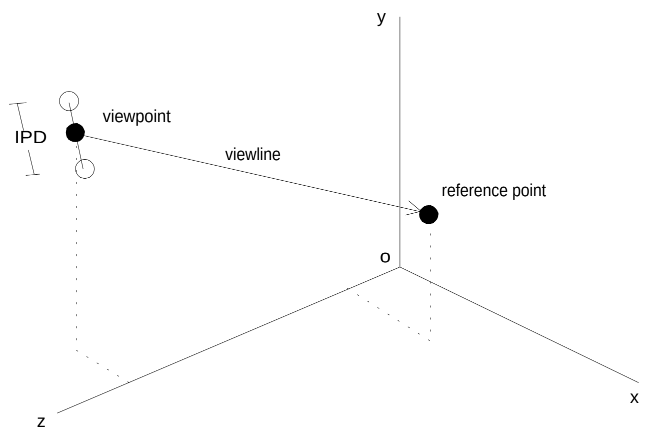

Figure 4.2: Initial viewer position and view direction

The available functions in the movemode:

middle button: leave movemode

left button: while the left button is depressed, moving the mouse has the following result:

- **up** move towards object
- **down** move away from object

no button: with no button depressed, the viewer moves as follows:

- **left** turn clockwise around object
- **right** turn counter-clockwise around object
- **up** turn upward around object
- **down** turn downward around object

Viewer left- and right turns are around a line parallel to the y-axis and through the reference point. Up- and downward turns are around a line through the reference point and parallel to the line through both eyes. Turns rotate the viewpoint with respect to the reference point. Moving forward or backward moves along the viewline. All possible movements are visible in Figure 4.3. Note that the eyes always move in a plane parallel to the XOZ plane. There is also a reduced movemode in which the viewer cannot turn. During moving the frame rate is measured.

Toggle look at origin

For some tests it is required that the viewer is looking exactly at the origin (i.e. the point with coordinates $(0, 0, 0)$), which is why this option was added. Using the GL `lookat` function this was easily implemented.

Toggle right image mirror

For a specific test it is required that *one* of the images can be turned 180 degrees upside down. Again this was implemented using the `lookat` function, which has an extra `twist` parameter that specifies the angle the eye is turned.

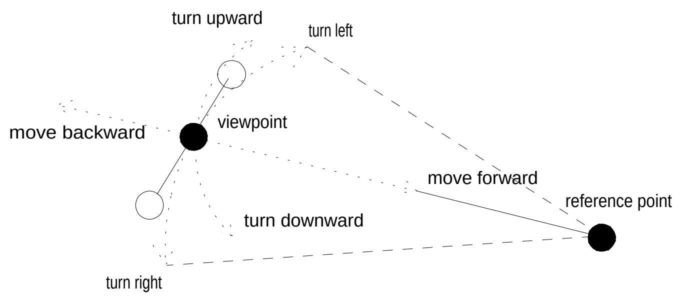

Figure 4.3: Possible movements

Chapter 5

Tests

5.1 Introduction

To assess the system's usefulness for conducting stereoscopic viewing tests, a test procedure was designed and twelve persons were tested. Section 5.2 shows that HMD screen resolution very much limits the best obtainable accuracy. Our test procedure is described in Section 5.3, followed by the results in Section 5.4.

5.2 Depth in an HMD

In Section 3.2 the relation between the best obtainable acuity and the amount of depth in an image was shown. The amount of depth was measured using *depth steps*: the number of distances at which objects may be positioned in a distance interval such that they are perceived as all being at different distances.

Given the following variables:

- $hres$: the horizontal resolution of an HMD display screen

- $width$: the width of this screen
- z_v: the distance of the virtual image of the screen

and the fact that smallest possible horizontal difference is of course one pixel, we can compute the best obtainable acuity in an HMD:

$$pixelwidth = width/hres$$

$$\alpha = \arctan(pixelwidth/z_v)$$

And the number of depth steps s beyond distance d:

$$s = \arctan(IPD/d)/\alpha$$

An example using the values of our configuration, an average IPD and the value of d that was also used in Section 3.2:

- $IPD = 0.065m$
- $d = 0.3m$
- $hres = 207$
- $width = 0.0542m$
- $z_v = 0.3982m$

$$pixelwidth = 0.0542/207 \approx 0.000262m$$

$$\alpha = \arctan(0.000262/0.3982) \approx 2.16'$$

$$s = \arctan(0.065/0.3)/2.16' \approx 324$$

Recall that the corresponding value for a stereo viewer was 490, and for real life 730. We may conclude that the amount of depth in an HMD is approaching that of a stereo viewer.

5.3 Test description

Our test procedure involved the following steps:

1. stereoscopic viewing test
2. IPD measurement
3. IPD test in the VE, using a specially designed test object
4. test using converging versus parallel viewlines
5. predistortion test using a regular grid

The system's settings for all tests were as follows:

- off center correction on
- initial IPD at least 10 mm above measured IPD
- converging viewlines (except in the parallel viewlines test)
- predistortion off (except in the predistortion test)
- measured axes distance and angle correct
- measured FOVs correct

5.3.1 Stereoscopic viewing test

Test persons must be able to view stereoscopically. If they are not, they are not allowed to do any further tests. The test we use is the *TNO test for stereoscopic vision* [IT72]. The viewer wears a pair of glasses with the left glass coloured red, and the right one green. Next a series of random dot stereograms is presented, containing pictures requiring a certain stereo acuity in order to be seen.

5.3.2 IPD measurement

The actual IPD of the viewer is measured, for comparison with the values that result from the following test. The initial IPD is set to at least 10 mm more than the measured value, to ensure that it is incorrect when an IPD test is started.

5.3.3 IPD test using special test object

The test object consists of the letters OXO, with a vertical line above the X in the left eye image, and a vertical line below the X in the right eye image, as can be seen in Figure 5.1.

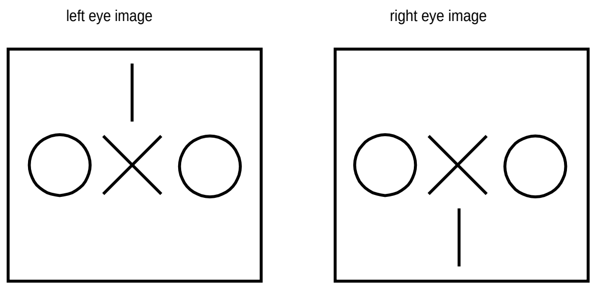

Figure 5.1: IPD test object

The fact that the left and right eye image are different is because the brain is very much able to correct for stereo images based on a incorrect IPD. The idea with this test object is that the brain will attempt to converge the central part of the images (the OXO), and probably succeed if the computational

IPD is not too far off, and leave the vertical lines at their true position, as they are each conflicting with the data received by the other eye. The result is that the viewer will change the computational IPD until the OXO converge, which typically happens when it is less than about 3 mm off from the actual IPD. The vertical lines however are usually not aligned by then, which makes a further, more precise adjustment of the IPD possible. Note that the usefulness of this test is solely based on the assumption that the brain behaves differently for different areas of a perceived image.

5.3.4 Test using converging versus parallel viewlines

From now on the IPD value is set to the value resulting from the test with the special ("OXO") test object, as apparently that is the value with which the viewer is most comfortable.

An ordinary object at correct scale (a coffee mug) is presented at close distance (0.40 meter) to the viewer using converging and using parallel viewlines. The viewer is asked which image is most comfortable to the eyes.

5.3.5 Predistortion test using regular grid

The viewer is positioned directly in front of a regular grid, and is asked if the grid lines in the edges of the image either:

- curve outward
- are straight
- curve inward

The predistortion coefficient is adjusted (in steps of 0.05) until the viewer is convinced that the grid lines appear straight.

5.4 Test results

Twelve persons were tested. Per test the results are given below.

- **Stereoscopic viewing test**

  Every test person was able to see stereoscopically. The average stereo acuity was 68.75" $\approx$ 1'.

- IPD measurement

  The average IPD was 65 mm.

- IPD test using special test object

  This test was conducted twice, once with the test object at a distance of two meters, once at a distance of 0.5 meters. The width of the test object is approximately 0.4 meters.

  The average IPD with which each test person viewed most comfortably was 63 mm with the test object at 2 meters, and 66 mm with the test object at 0.5 meters.

  A suitable way to conduct this test is to set the initial IPD larger than the measured IPD. In this way we are sure that the viewer perceives an incorrect image, because he cannot diverge his eyes. The viewer is then instructed to decrease the IPD until the image of one three-dimensional object appears, which can be comfortably viewed.

  The significance of these results is reduced because the precise mapping of the images on the operator screen to the display screens inside the HMD is not known. For this the HMD must be disassembled. Already by looking with the right eye through the left eye optics and vice versa, it could be seen that:

  - not all of the operator screen images is visible on the HMD screens
  - the loss of data is different for each side of a screen
  - the loss of data is different for the left and right screen

This implies that all correctional translations are inaccurate: they are calculated in operator screen pixels, assuming a certain number of pixels map on a certain width in millimeters on an HMD screen.

- Test using converging versus parallel viewlines

  Nine out of twelve test persons (≈ 75 %) judged the images presented using converging viewlines more pleasant to watch. Three test persons liked the parallel viewlines version better. Theoretically the images generated using parallel viewlines were the least pleasing to the eyes, because the object viewed was positioned at a distance of 0.4 meters. This caused distortion in the parallel viewlines images.

- Predistortion test using regular grid

  The theoretically optimal predistortion coefficient is -0.18 [RR91]. The average of the coefficients chosen by our test persons to be optimal is -0.17125 ≈ -0.17.

  The difference between our value and the "optimal" one may be explained by two reasons:

  - as has been stated above, the exact operator screen image to HMD screen mapping is not known, causing pixel positions calculated in the precomputing stage to be slightly off
  - when someone focuses at a certain distance, his "focal plane" is slightly curved, implying that a grid displayed as in our test perhaps must also be slightly curved to *appear* straight. Such a grid would then be "straighter than in reality", however.
  - before the grid is predistorted, the viewer sees a (pin-cushion) distorted grid. This may influence the viewer such that he sees a barrel-distorted grid after predistortion, even if this grid is actually straight.

Chapter 6

Discussion and conclusion

Before the beginning of this project, three goals were formulated:

- identification of parameters influencing depth perception in an HMD
- development of a test system allowing independent manipulation of all parameters
- implementation of a test enabling setting of user-dependent parameters inside a Virtual Environment

It follows from this report that those three goals were achieved. The fact that all HMD specific parameters are maintained in a separate file enhances the system's flexibility.

With respect to the first goal, identification of the parameters, it is interesting to attempt to determine the relative importance of each parameter.

We believe that the most fundamental requirement of stereoscopic images generated by an HMD is that the eyes of the viewer are able to converge on the images, i.e. enable the brain to combine them to *one* image. The horizontal position of the images, the type of projection used and the angle between the optical axes all influence this ability. Assuming that aberrations are all of the same order, we arrange them as follows:

1. horizontal position, which in turn depends on the
   - IPD
   - screen center offset
   - distance between optical axes
   - distance between viewer and object [1]
2. angle between optical axes. Very small angles (< 2 degrees) may be tolerable.
3. projection type. We found that projection without converging viewlines is especially disturbing for viewing at close distances. Assuming a VE application has one object of interest (e.g. a tool) at a small distance, one may choose to render this object using converging viewlines (= off-axis projection), and all other objects (the surroundings) using on-axis projection. A projection using parallel viewlines is not recommended: it always produces distortion, depending on the object distance.
4. optics distortion

   The optics distortion may also result in incorrect convergence, especially near the edges of the image. Apart from that it obviously causes the image to be incorrect. The importance of this error also very much depends on the VE application type. Currently the predistortion is computationally too expensive to use it during real-time rendering. In our system, after optimization and parallel implementation, a frame rate of 4 Hz may be achieved. The predistortion can easily be implemented in hardware, which should be done if real-time rendering is required.
5. Field Of View

   As an incorrect FOV only results in a size error, we classify it as the least important error. Naturally this may not be the case, depending on the application requirements.

[1]The last of course only when a projection type is used that requires a horizontal translation to get the convergence right.

Note that the resolution of the HMD display screens determines whether a certain error correction is useful or not: if a positional error does not cause a shift of at least one pixel, it will not be visible anyway.

Concerning the special IPD test object, it will be interesting to see if after incorporation of the exact operator screen images to HMD mapping the test can be used to determine *one* comfortable IPD for all distances. In our opinion the test object should be as large as possible, so it does not become too easy for the brain to combine both images.

Another improvement would be to include eye trackers inside the HMD. These will have to be accurate enough to be able to determine on which object the viewer is focusing.

Suggested extensions to the SOS are:

- parallel implementation of the predistortion
- incorporation of the position- and orientation sensor data
- combininig projection types in one rendering

Also parts of the SOS could be ported to the dVS VE operating system running on our workstation.

To summarize, we recommend the following setup:

- ensure correct horizontal position of the images, a correct computational FOV and optical axes angle
- use converging viewlines for nearby (closer than 3 meters) objects, and on-axis projection for far objects
- use predistortion implemented in hardware

After taking these measures, we will come close to achieving *orthostereoscopy*.

Appendix A

Implementation aspects

This appendix is intended for readers who want to acquire insight into the program structure of the SOS.

General information

The system has been written in C++, using the Gnu gcc compiler version 2.3.3, the GL graphics library and the Forms user interface library [Ove92]. Not counting the system, GL and Forms Library header files, the total source length is about 5300 lines.

Program decomposition

Recall the class hierarchy of "has-a" relations resulting from the design phase (see Figure A.1).

The following "main" classes are defined: **operator, menu, viewer, view, hmd, images, object, predist, videosplitter, status, syspar**. Supporting classes are: **datafile** (general ASCII token file), **Window** (standard GL window), **TextWindow** (derived from Window, adds text routines), **GraphicsWindow** (derived from Window, adds graphics routines), **slider**

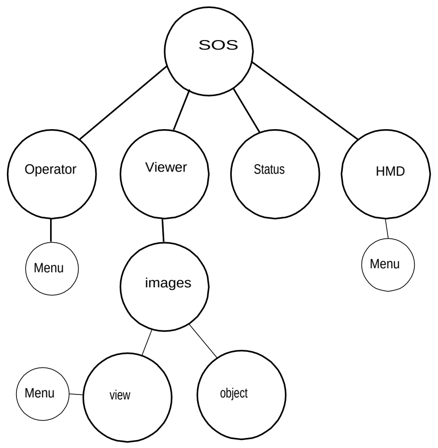

Figure A.1: Object Relationship Graph

(general slider), **mouse** (mouse routines), **picklist** (file picklist), **light** (object lighting). Miscellaneous source files are: **misc** (some conversion functions, e.g. `strupr`), **vector** (vector functions), **sysdat** (global declarations: e.g. colours), **error** (error message function).

The window classes are all contained in `window.cc` and `window.hh`. The hmd class source is split into four files: the header `hmd.hh`, and `hmd.cc`, `hmdfile.cc` and `hmdcalc.cc`. The PAZ grammar is defined in `pazdat.hh`, the hmd file grammar in `hmddat.hh` and the viewer file grammar in `vwrdat.hh`.

Cross reference

In Table A.1 a cross reference is given between all main classes.

these ↓ use →	**view**	**viewer**	**hmd**	**predist**
menu	do_view_menu	do_viewer_menu do_view-point_menu	do_hmd_menu	
viewer	get_converging			
hmd				
images	get_predistortion	look	get_current_FOV	predistort
object		calculate_viewpoint		
view				
videospl.	get_converging get_off_center		get_center_offset get_screen_width	

these ↓ use →	**object**	**images**	**videospl.**	**status**	**datafile**
menu	do_object_menu	redisplay rotate_images setup_viewer		redisplay	
viewer	get_reference points	redisplay setup_viewer		*various*	*various*
hmd		redisplay setup_projection setup_viewer	get_resolution	*various*	*various*
images	render		get_origins get_resolution set_image_origins		
object		redisplay setup_viewer		*various*	*various*
view			set_image_origins	*various*	
videospl.		get_dimensions		*various*	

Table A.1: Method (function) calls between classes

Appendix B

Figure sources

Figure 3.2 [Fer87]

Figure 3.4 [VSLC90]

Figure 3.7 [RR91]

Figure 3.9 [How91]

Figure 3.12 [RR91]

Figure 4.1 [Atk92]

Appendix C

Acknowledgements

This project has greatly benefitted from the support of many people working at the Physics and Electronics Laboratory. I very much appreciate their help, which has been a main factor in finishing this project at the projected date. I specifically want to acknowledge:

- everyone working in the High Performance Computing group
- dr. G.J. Jense
- ir. F. Kuijper
- J. Stevens
- dr. P. van Oosterom
- everyone volunteering to be tested

And from outside FEL-TNO:

- dr. D.P. Huijsmans of Leiden University
- J.P. Rolland of the University of North Carolina at Chapel Hill
- E. Yeaman of Virtual Research

Bibliography

[Atk92] Phil Atkin. Paz object file specification. Technical report, Division, 1992. appendix K to the dVs v0.2 documentation.

[Fer87] J.G. Ferwerda. *A practical guide to stereo photography.* 3-D Book Productions, 2nd edition, 1987.

[HN92] Mats Henricson and Erik Nyquist. Programming in c++, rules and recommendations. Technical Report M 90 0118 Uen, Ellemtel, 1992.

[Hod92] Larry F. Hodges. Time multiplexed stereoscopic computer graphics. *IEEE Computer Graphics and Applications*, 12(2):20–30, march 1992.

[How91] Eric M. Howlett. Wide angle orthostereo. In *Proc. SPIE vol.1457, Stereoscopic Displays and Applications II*, pages 210–223, 1991.

[IT72] IZF-TNO. *TNO test for stereoscopic vision.* Laméris Ootech, ninth edition, 1972.

[Min93a] P. Min. Stereoscopy optimization system, design and specification document. Technical report, FEL-TNO, May 1993.

[Min93b] P. Min. Stereoscopy optimization system, requirements definition document. Technical report, FEL-TNO, March 1993.

[Ove92] Mark H. Overmars. *Forms Library, A Graphical User Interface Toolkit for Silicon Graphics Workstations.* Utrecht University, the Netherlands, 2.1 edition, 1992.

[RR91] Warren Robinett and Jannick P. Rolland. A computational model for the stereoscopic optics of a head-mounted display. In *Proc. SPIE vol.1457, Stereoscopic Displays and Applications II*, pages 140–160, 1991.

[Sut68] Ivan E. Sutherland. A head-mounted three dimensional display. In *Proc. Fall Joint Computer Conference*, pages 757–764, 1968.

[Ter86] Patrick D. Terry. *Programming Language Translation.* Addison-Wesley, 1986.

[Val66] N.A. Valyus. *Stereoscopy.* Focal Press, London, 1966.

[vdB88] J. van den Bos. Design and specifications based on a protocol-constrained object language. Technical report, Leiden University, January 1988.

[VSLC90] Harry Veron, David A. Southard, Jeffrey R. Leger, and John L. Conway. Stereoscopic displays for terrain database visualization. In *Proc. SPIE vol.1256, Stereoscopic Displays and Applications*, pages 124–135, 1990.

[Win92] J.F.H. Winkler. Objectivism: "class" considered harmful. *Communications of the ACM*, 35(8):128–130, August 1992.

[WP90] Steven P. Williams and Russell V. Parrish. New computational control techniques and increased understanding for stereo 3-d display. In *Proc. SPIE vol.1256, Stereoscopic Displays and Applications*, pages 73–82, 1990.

[WS82] Günter Wyszecki and W.S. Stiles. *Color Science, Concepts and Methods, Quantitative Data and Formulae.* John Wiley & Sons, 2nd edition, 1982.

[You91] Yourdon. *Structured Analysis for Real Time Systems, Course Lecture Notes*, 4.1 edition, 1991.