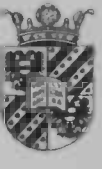
University of Groningen **Faculty of Mathematics and Natural Sciences**

Department of Mathematics and Computing Science

Colour perception in a virtual scene

Romke Dam



Supervisors:

dr. F.W. Cornelissen, drs. T.L. Vladusich

Laboratory for Experimental Ophthalmology

prof.dr. J.B.T.M. Roerdink

Department of Mathematics and Computing Science

December, 2005

RuG

Colour perception in a virtual scene

Romke Dam

December 20, 2005

Abstract

Why do we always perceive the colour of grass as green, even under changing daylight conditions? From a physical point of view, grass is not always green. The light illuminating the grass varies throughout the day due to atmospheric wavelength scattering and refraction. The ability to perceive constant colours under changing illumination conditions is called colour constancy. An important unanswered question regarding colour constancy concerns the role of binocular information. Binocular disparity facilitates three-dimensional (3-D) object perception, and, in principle, may perform a role in colour constancy.

In this master thesis, we focus on the influence of binocular information in the perception of surface colour in a virtual 3-D scene. In particular, we examine whether the addition of binocular disparity cues may modulate the extent to which nearby surfaces within a two-dimensional (2-D) projection plane of the virtual scene affect the colour of a target surface (colour induction, a phenomenon closely related to colour constancy). Consistent with previous studies, we find some evidence to support the notion that colour induction is strongest when object surfaces appear in the same depth plane. We discuss technical problems associated with developing the virtual scene and suggest improved experimental designs for investigating additional questions related to colour constancy and depth perception.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Although scientists have studied visual perception in humans and other animals for over a century, a clear understanding of it is still lacking. One of the best-studied aspects of vision is the phenomenon of colour constancy, which refers to the fact that objects' colours (e.g., grass) tend to appear the same despite changing illumination conditions (e.g., red sunsets). Research into colour constancy involves identifying the importance of different perceptual cues [17, 10]. One cue that may play a role in colour constancy is binocular disparity [18], which facilitates our perception of the three-dimensional (3-D) world.

While binocular disparity is a cue that is present in all visual displays that investigate colour perception in real environments [6], the growth in computer graphics technology allows researchers to quickly and easily manipulate binocular disparity in virtual environments [3]. In the present research, we have investigated whether the perceived depth of objects surrounding a target surface affects the perceived colour of the target in virtual environments. From this experiment we may find out whether binocular disparity can contribute to aspects of colour perception. (In this case we examine the case of colour induction, which is closely related to colour constancy. We also examine the strength of colour constancy in our experiment, although we do not manipulate depth cues in that condition.)

In the following chapters we will introduce basic aspects of human vision (Chapter 2). We then introduce the concepts of colour constancy and binocular depth perception in further detail (Chapter 3), leading directly to the experimental questions. We then explain the methods to conduct the experiment (Chapter 4). We then present the results (Chapter 5) and discuss their meaning and potential for further experiments (Chapter 6).

Chapter 2

Human perception

Seeing is the ability to perceive electromagnetic waves by our eyes and transforming these into a visual representation. This representation is a result of psychophysical and physiological processes. The created image is formed by the response to lightwaves that hit our eyes. Light is the result of packets of highly localized energy that we call photons. A photon, in turn, is a single packet of light that can be viewed as a wave pattern and a particle that carries a certain amount of energy. The wavelengths to which the human eye is sensitive range from 380 to 750 nm and the characteristics of colour perception are determined by these wavelengths. We cannot see a single photon and visual perception is based on vast amounts of photons entering the eye simultaneously. The eye thus receives large amounts of different lightwaves, which are transformed into electrical signals that are sent to and further processed by our brain. In this chapter we will discuss the workings of the human eyes and how they help us perceive our surround.

2.1 Journey into the eye

The eye can be divided into different parts where each part plays a specific role. The eye is not a static organ. It is located in the eye socket, or bony orbit and is placed close to the brain at a relative high point on our bodies and the eye points slightly outward. This allows us get a good view of our environment and fast transmission of signals to our brain. The eye is dynamic; muscles attached to our eyes make them active receptors that make it possible to concentrate on and track objects. Without being aware of it, the eye makes small, involuntary movements of about 3Hz, called mini-saccades. Without these small movements we cannot maintain vision and our visual world would quickly fade away.

Light which enters our eyes passes through the cornea, which is a transparent complex lamellar structure. Blinking helps us to keep the cornea clean thereby maintaining its optical interface. The cornea is a primary refractive element of the eye, because its index of refraction is greater than that of air. The smoothness and degree of refraction make the cornea important in the way light enters our eye. The iris and pupil regulate the amount of light entering the eye. The iris controls the entry of light into the eye and the pupil determines the amount of light that can pass to the retina without altering the field of view. The iris also determines the colour of the eye.

The quality of the retinal image depends on the size of the pupil. The difference of a small pupil in contrast to a large one is the depth of focus. Reducing the size of the pupil narrows the bundle of light hitting the retina and thereby decreases the blur of the image. lightwaves that can pass through the iris and pupil move on to the lens. The lens is a complex multi-layered structure

which has the ability to change during accommodation. The process of accommodation helps us to perceive a clear image of objects at various distances. The lightwaves that are passed through the lens fall onto the retina, a multi-layered sensory tissue that is located at the back of the eye. The top layer of the retina is covered with photosensitive cells. These cells convert light energy into electrical signals that are carried to the brain by the optic nerve.

2.1.1 Cones and Rods

Photosensitive cells in the retina convert incident light energy into signals that are passed through a network layer in the retina formed by the ganglion cells. The axons of these ganglion cells make up the optic nerve, the single route by which information leaves the eye. There are two types of photosensitive cells: rods and cones. The rods are primarily used for vision under low levels of brightness. The cones on the other hand are used in situations with high levels of brightness (like daylight) and make it possible to distinguish colour and see detail. The most important part of the retina concerning colour vision is the *fovea centralis*. The fovea is the centre of the retina and is made up of a high density of cones (in a hexagon-like pattern) and is, therefore, the region of highest visual acuity. Humans tend to direct their gaze at the object under study such that light from this object reaches the retina at the fovea. All these accounts make the fovea largely responsible for the sharp colour vision in humans.
Largest part of the approximately six million cones is located on or near the fovea. Humans have three types of cones, which can be classified as the long wavelength (L), medium (M), and short(S) wavelength cones, which respond to yellowish-green, green and bluish-violet light (peak wavelengths of 564 nm, 534 nm, and 420 nm, respectively; see Figure 2.1). The eye is more sensitive to green light than other colours: when looking at a green light and a red light of the same intensity, the green light will appear to be brighter. The S-cones are sensitive to light at wavelengths shorter than 400 nm, but the lens and cornea of the human eye are increasingly absorptive to these wavelengths and this sets the lower wavelength limit of human-visible light to approximately 380 nm (which is the onset of ultraviolet light). It is important to note that the LMS system provides us with knowledge about the sensitivity of each cone type for each wavelength and can help in classifying differences or equality between different colours, but does not give us a proper insight on the actual *appearance* of a colour we perceive.

2.2 Colourimetry

Although we have extensive knowledge of the connections between the eye and the brain, we do not know all details of the different processes in the eye and brain that are involved in the transformation from physical light to subjective percepts. Colour perception is only one aspect of visual perception and can be viewed from different angles. We can, for example, distinguish between the physiological and psychophysical response to certain stimuli. If we look at the physiological response to a certain light stimulus, we look at the different biochemical processes in the eye and brain involved in processing these stimuli. Psychophysics is concerned with the behavioural response to physical stimuli. While the physiological processes can be defined in physical terms (for instance cone stimulation), this is difficult for psychophisycal processes, because these deal with the subjective percept that results from a stimulus. Therefore, colour in the psychophysical sense refers to an abstract sensation of colour.

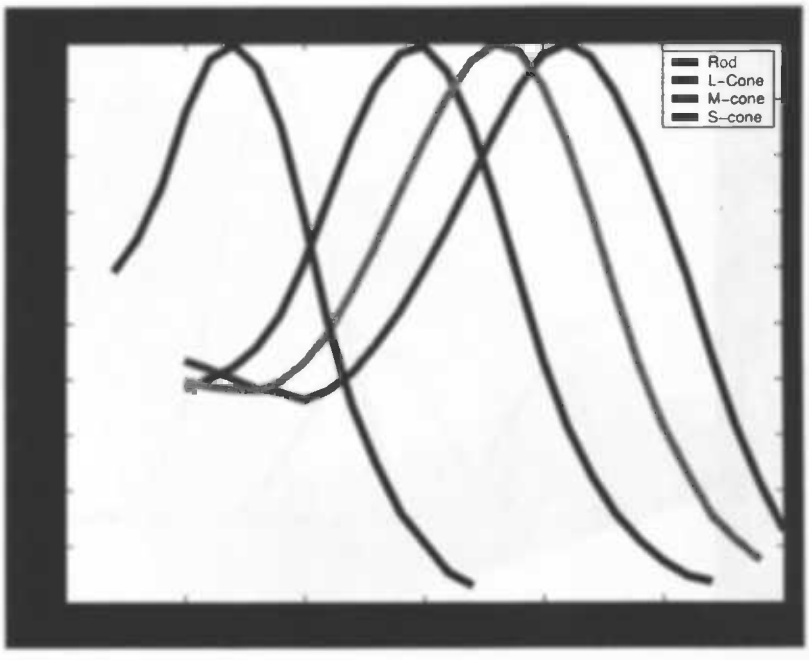


Figure 2.1: *L,M and S and rod relative sensitivity (Dartnall, Bowmaker & Mollon 1983).*

Each field has its own interpretation of colour. For artists, colour means the reflectance property of paint, while an architect looks at colour in the context of the properties of involved light sources. Consequently, all fields that use the term ,,colour" use it with a different meaning. Because colour is a highly abstract entity, it is hard to transform these meanings into physical values.

Colourimetry is the science and technology used to quantify colour and describe human colour perception in physical terms. In 1931 the CIE (Commission Internationale de l'Eclairage or International Commission on Illumination) looked for a way to set a standard in colour perception and made the first steps in colourimetry. To set a standard, the CIE determined average results of colour matches using three spectral primaries for a large number of observers. These observers viewed a 2° circular split field, which is the angular size of the human fovea, and their task was to adjust the three primaries so that their chosen mixture would visually match the visible spectrum of the presented stimuli. The colour match function can be described as: $c(C)+r(R) = g(G)+b(B)$. Here C units of the test colour c plus R units of the primary red (r) additively mixed should match G units of the green primary (g) plus B units of the blue primary (b). The results of a similar type of experiment can be seen in Figure 2.2.

Different mixtures of colours can appear identical to a human observer. This is a phenomenon called metamerism, which was one of the ideas behind the matching experiment. We must point out here that these equivalences are based on empirical settings rather than on mathematical or physical equality. But, if the analogous mathematical statement is written and manipulated and in accordance with the rules of algebra, then it can be found experimentally that such calculations predict the result, within a fairy wide limit, of new colour matches when transformed back into the experimental matching function. The results from these matching functions were summarised as the standardized 1931 CIE RGB matching functions.

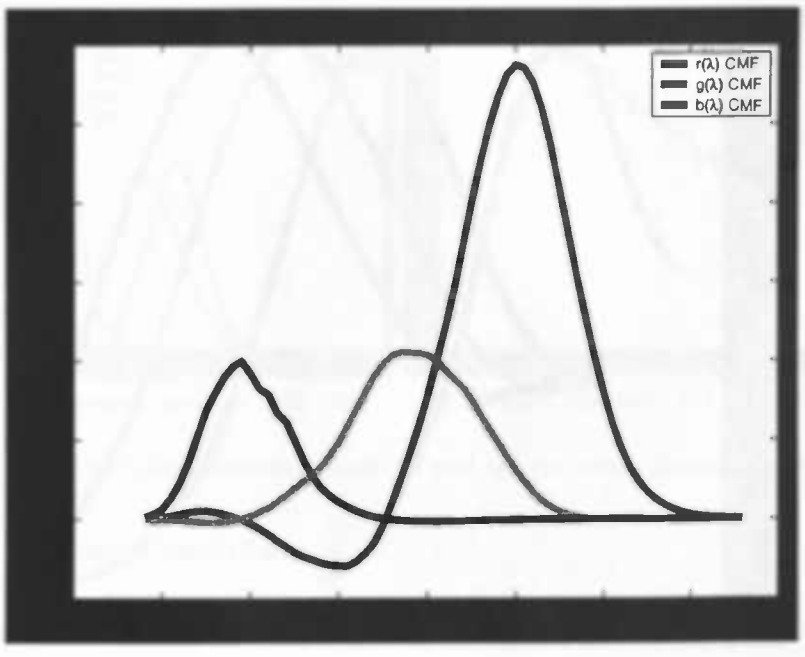


Figure 2.2: *Colour matching functions (Stiles & Burch 1955).*

2.2.1 CIE Tristimulus system

Because the RGB matching primaries did not account for the whole range of colours that humans can see, it had to be modified by allowing negative values in one of the primaries. This was considered a drawback. The CIE came up with an alternative to the RGB matching primaries that would compensate for these negative attributes and introduced the CIE XYZ system. A note must be made here that at the time the CIE created the XYZ tristimulus values little was known about the cone sensitivity of humans.
The X, Y and Z are imaginary primitives that were chosen in a manner so they could encapsulate the entire human colour space. The primitives can only have positive values and the Y primitive represents the brightness or luminance of a certain colour, which makes the primaries abide the laws of additive colour mixing. Brightness stands for the intensity of a colour and is measured in Candela per square metre, where a colour with zero intensity is black. The tristimulus values make up a canvas for the human perception space. White was chosen to be represented by equal X, Y and Z values. Because of these qualities the CIE could account for all the behaviouristic properties of colours such as chromaticities (*chroma* is the purity of a colour). Chromatic attributes can be found by calculating the projective coordinates of a tristimulus value. The projective coordinates (x,y) on the chromaticity diagram (Figure 2.3) occupy a region of the real projective plane. Although the projective coordinates have three dimensions, we only consider the x and y values because the sum of x,y and z must equal one (coordinates that do not meet this constraint represent colours that are not visible by humans and therefore fall outside CIE space). Therefore coordinates are usually plotted on a two dimensional chart, omitting the z value.
Achromatic colours are represented in the centre of the diagram; the more we move towards the borders, the more chromatic the colour becomes. The border of the chromaticity chart is formed by all the lightwaves a human can perceive. Thus the border can be viewed as the change in hue and

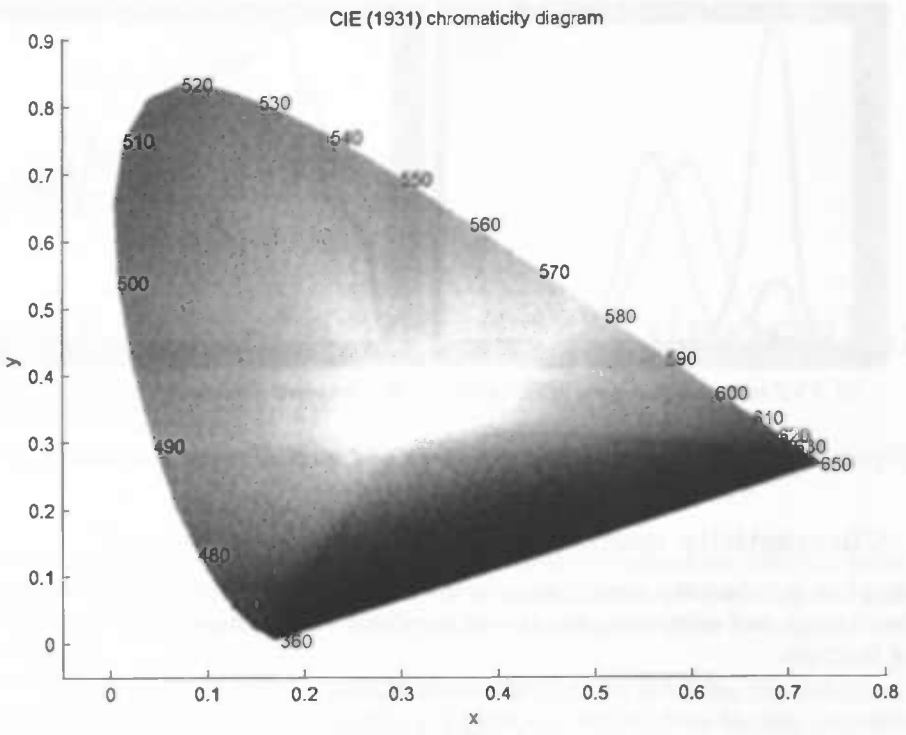
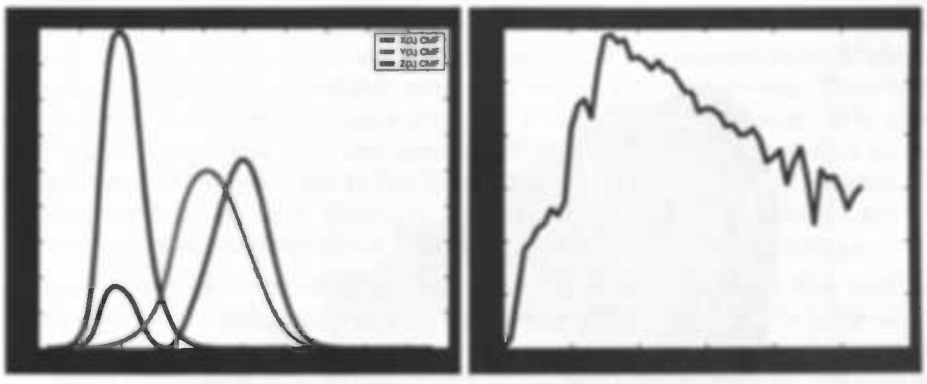


Figure 2.3: *CIE 1931 chromaticity diagram, a two dimensional space of all the colours humans can see. (Note that this figure is an approximation, because (all) graphical devices can only reproduce a subset of the diagram)*

moving from centre to border as the change in saturation. The chromaticity of a colour does not say anything about the physical properties of a colour but only about perceptual properties and helps to qualify (perceptual) differences between colours. Over the years, the CIE graph was altered to deal with some drawbacks and faults in the CIE XYZ scheme. One of these changes addressed the perceptual non-uniformity in the CIE 1931 colour space and from this the CIE Lab and CIE Luv were created. In these charts equal distance between points represent equal (perceptual) difference in chromaticity and/or luminance, making a display between different colours more intuitive and easier to calculate. But due to its wide-spread acceptance and use, even today colour is often expressed in CIE 1931 chromaticity coordinates.



(a) *XYZ matching functions (CIE 1931)* (b) *Standard Illuminant D65 (CIE 1931)*

Figure 2.4: *Tristimulus matching functions and spectral power distribution of D65*

2.2.2 Chromaticity coordinates

Computing the chromaticity coordinate of a given radiant object is done by taking its spectral power distribution and calculating the area of the distribution subjected to each XYZ tristimulus matching function.

$$\begin{aligned} X &= \int E_\lambda \bar{x} d\lambda \\ Y &= \int E_\lambda \bar{y} d\lambda \\ Z &= \int E_\lambda \bar{z} d\lambda \end{aligned} \tag{2.1}$$

Here E is the radiant energy at lightwave λ and $\bar{x}$, $\bar{y}$ and $\bar{z}$ are the matching functions. From these values we can calculate the chromaticity coordinates in the following way.

$$\begin{aligned} x &= \frac{X}{X+Y+Z} \\ y &= \frac{Y}{X+Y+Z} \\ z &= 1-x-y \end{aligned} \tag{2.2}$$

In the case of the illuminant D65 on Figure 2.4, which closely resembles the relative spectral energy distribution of north-sky daylight, its chromaticity coordinates will be: *x 0.313 y 0.329*, which can be interpreted as a achromatic or "white" colour.

Chapter 3

Colour constancy and binocular perception

While colour constancy and binocular perception appear as two extremely different fields of human vision, they are actually closely related. The first gives humans the ability to create a sense of spatial surrounding; the second ensures a sense of constant colour perception under global illuminant changes.
The human visual system is a complex and sophisticated system involving physiological, biochemical, neurological and psychological processes. To get a grasp on how human perception works, perception is modelled on the processing of information cues. These cues are viewed as being part of a network where every input can influence other cues; if cues contain false information they can lead to illusions in perception. Vision is not a passive "photographic" recording of surrounding objects. Instead, it is the active process of perceiving those objects. One mechanism of active recording of the environment is depth perception, however for most humans depth perception is a subconscious process.
The first major breakthrough in stereopsis was in 1830 through a publication in the Royal Society of London. In a classical article written by Charles Wheatstone [14] the perception of binocular disparity is studied by the use of stereo grams. With the aid of a self-created stereo setup he revolutionised all former ideas about perception. In the article he presents different experiments, in which he discovered that both physiological and psychological aspects play a major role in stereopsis.

3.1 Binocular perception

Since the publication of Charles Wheatstone much progression has been made in the field of stereopsis and his discoveries are still applied in modern technology. Depth perception can be categorised in ten cues [8, 12]. We will first describe the cues in which the eyes plays a significant (active) role, followed by a presentation of cues that are purely monocular.

- *accommodation*, the adjustment of the focal length of the lens.
- *convergence*, the angle made by the two viewing axes of a pair of eyes.
- *binocular disparity*, the disparity between images of the same object projected onto the retinas

Accommodation and convergences are considered to be mirror cues in depth perception, as they are associated with the eye muscles, and interact with each other. Accommodation is also considered a monocular depth cue since it is available even when we see with a single eye. This cue is effective only when combined with other binocular cues but is considered quite weak. It is effective only at short viewing distances (less than two meters). When watching an object close to us, our eyes point slightly inward. This difference in the direction of the eyes is called convergence. This depth cue is also effective only on short distances. Convergence and accommodation are oculomotor cues, they involve getting proprioceptive feedback from the muscles of the eyes.

Binocular disparity is considered the most important depth perception cue over medium viewing distances. It refers to the difference between the images of the same object projected onto each retina. The degree of disparity between the two images depends on the parallactic (convergence) angle, because the eyes are separated horizontally by the interocular distance. This angle is formed by the converging of each eye on an object. The parallactic angle is related to the distance of an object from the eyes. At great distances the parallactic angle decreases and depth perception becomes increasingly difficult.

The monocular cues are: *retinal image size*, *linear perspective*, *aerial perspective*, *overlapping*, *lighting and shade (shadows)*,*texture gradient* and *motion parallax*. Retinal image size can be used as a cue when the real size of the object is known and our brain compares the sensed size of the object to this real size, and thus acquires information about the distance of the object. If we see a lion and a mouse being the same size on the retina we expect the lion to be further away. A guideline for a known object is, the larger the object image the closer it is.

When looking down a straight level road we see the parallel sides of the road meet in the horizon and this is termed linear perspective. The converging parallel lines indicate increasing distance. Linear perspective is considered an important depth cue.

Aerial perspective is a cue that interprets the haziness of distant objects as depth information; light coming from a distant object will have some of its reflected light scattered. This will have an effect upon the light reaching our eyes. Since all the lightwaves reflecting from the distant object are not travelling in a straight line, we see the distant objects a bit fuzzy but still in focus.

The effect where continuous outlines appear closer to the observer is called overlapping. When objects block each other out of our sight, and we know that the object that blocks the other one is closer to us, the object whose outline pattern looks more continuous is felt to be closer to the observer.

Also lighting, shade and shadows can contain depth information. Closer objects are brighter, while distant ones dimmer. There are a number of other more subtle cues implied by lighting, for example, the way a curved surface reflects light suggests the rate of curvature, and shadows are a form of occlusion.

Texture gradient is a type of linear perspective relating levels of roughness of a uniform material as it recedes into the distance. The closer we are to an object the more detail we can see of its surface texture. Objects with smooth textures are usually interpreted as being farther away. This is especially true if the surface texture spans the entire distance from near to far.

Visual motion parallax is a function of the rate at which the image of an object moves across the retina. Objects that are in the distance will appear slow in comparison with objects closer to the observer even when the two are moving at the same speed. Motion parallax can also be caused by movement of the viewer's head.

3.2 Colour constancy

The relation between colour constancy and depth perception can be seen as the inner workings between perceiving spatial surround and the illumination present in that scene. The interaction between light and objects helps us create a sense of space. Colour constancy involves a process of estimating the global illumination. Simplifying the scene from an observer's point of view, we can model the perceived scene as a colour signal. The colour signal consists of the lightwaves reflected to the eye from each visible scene location. We can model the light hitting the eye in the following way [4]:

$$C(\lambda) = S(\lambda) \times E(\lambda) \tag{3.1}$$

Light (E) that scatters uniformly in a scene can be viewed as a spectral power distribution. This function specifies how much power the illuminant contains at each wavelength(λ). The objects (S) we see are the reflected rays bouncing off an object to the observer's eye (C). The light, i.e., the spectral reflectance, specifies what fraction of incident illumination is reflected from the object at each wavelength. The spectral power distribution of the colour signal is readily calculated from the illuminant spectral power distribution and surface reflectance function. Under different global illumination the spectral power distribution changes, therefore altering the colour signal.
Numerous cues allow us to distinguish between the effects of (a) the illuminant and (b) the reflectance properties of objects on the perceived colours. This distinction made it possible to suppress (global) illuminant changes. Thus, perfect constancy would occur if we perceive the same colour when the spectral power distribution changes. Therefore colour constancy can be seen as mechanism where changes to the spectral power distribution have no influence on the colour signal.

3.3 Colour constancy and binocular perception.

Can binocular disparity cues affect colour perception? In particular, we examine whether the presence or absence of binocular disparity can affect the amount of colour induction observed in a surface. Colour induction is the apparent change in surface colour caused by the presence of spatially adjacent colours. A classic example of colour induction is a physically grey patch on a green background: the patch appears coloured with complementary hue to the surround. In this case, the patch would appear more reddish. Importantly, the mechanism underlying colour induction is likely to be the same mechanism responsible for colour constancy [13, 7].

In a virtual scene created using the procedures documented in Chapter 4, we design an experiment in which the surface colours immediately adjacent to a target surface (adjacent in the sense of the 2-D projection of the 3-D scene) are made to lie within the same depth plane as the target, or a different depth plane. The aim is to test whether colour induction is stronger when the target and surrounding surfaces lie in the same depth plane. A previous study suggests that luminance perception (or achromatic colour perception) may depend on the 3-D layout of the scene [16], with greater induction observed for co-planar surfaces.

Chapter 4

Methods

In this chapter we will describe the experimental setup. We will first discuss virtual environments. Thereafter, we discuss the hardware that we used to create the virtual environment for our experiment, followed by a description of the software. Thereafter, we discuss the experimental design.

4.1 Virtual environments

Virtual environments are computer-generated sensory inputs that deceive a person by letting him experience a non-existent reality as real. What a virtual environment thus does is to mimic the sensory inputs that a person would receive when he[1] was in a real version of the simulated environment. Clearly, the more accurate this mimicking occurs, the more real the experience. Due to the extremely complex nature of the sensory inputs sent to us by real environments, it is very difficult to achieve a perfect result with virtual environments. One of the problems is computational constraints. Virtual environment stimuli often have to be generated in real-time, with 25 or more frames per second. Consider for example the light that arrives at our retinas. This travels with a speed of 300.000 $\frac{m}{s}$ and changes its characteristics each time it hits a material (filtering out of light with particular wavelengths). It is simply impossible to simulate this phenomenon at real-time on today's computers. Another problem is that for perfect realism we do not only have to simulate all signals that reach our senses, but also the physics of the entire (perceptible) environment. For instance, if we want to create a realistic virtual forest we need to model every single leave, branch and trunk and the dynamics (think of wind) for all the trees present in the scene. Rendering all the polygons and light interactions is a time consuming task. Fortunately, we have come a long way already and advances in computer graphics and technology are still making leaps forward. This in combination with the innovation of new algorithms and hardware, approximations of reality keep on getting better. If we compare modern day graphics to the graphics at hand a decade ago the differences in quality and performance are astonishing.

[1] Everywhere in this document 'he' should be read as 'he or she' and 'him' as 'him or her', etc

4.2 Hardware

4.2.1 Assessment of several VE setups

There are several types of setup for creating virtual environments. One option is to use the Reality Cube (CAVE) of Groningen's Centre for High-Performance Computing & Visualisation (HPC&V). The Reality Cube is an active stereo setup. Stereo images are presented by four screens in a box setup surrounding an observer, where he perceives these stereo images with the aid of active shutter glasses. While this setup fulfills the stereoscopic needs, it has a considerable drawback with respect to the luminance level of its output. Due to the particular type of projectors used and the stereoscopic glasses, the light eventually entering the eye has very low intensity (4 $\frac{cd}{m^2}$). This was unacceptable for us, since for our experiment it was crucial that vision was for the greatest part based on cone stimulation, while in dim environments rods take over vision.

An alternative would be to make us of the HPC&V's so-called Powerwall. This setup is a passive stereo setup. Stereo images are produced by two DLP projectors with a polaroid filter on each lens. These images are presented on a single screen, where an observer achieves stereo perception with the aid of polaroid glasses. This setup has the necessary luminance levels but due to the non uniformity (hotspot) and unpredictability (due to a 'white filter' in the DLP projectors) of the output, this setup was also rejected as a candidate for our setup experiment.

Because of the problems with the Reality Cube and the Powerwall, we chose to construct a so-called Wheatstone setup [14, 15]. This is a fairly simple setup that can be constructed by means of two monitors and two mirrors.

4.2.2 Our Wheatstone setup

There is a vast range of different (manufactured) Wheatstone setups. The differences between these setups concern the components and the purpose they are used for. Setups range from a plastic handheld device to large steel constructions.

The perceptual quality of the output of a Wheatstone setup depends on a variety of factors. First of all the mirrors that are used are important; if the quality of reflection by the mirrors is bad, the quality of the input to observers' eyes will be bad as well. Bad mirrors do not perfectly reflect lightwaves at the angle of incidence, which creates a scattered reflection. As a result, observers will see some blur in the images and some objects may have ghost fragments. In addition bad mirrors can cause chromatic aberration (light of different wavelengths being reflected in different angles). To obtain the best possible results, front surface mirrors should be used in our experiment

Another quality factor of a Wheatstone setup concerns the quality of the stereoscopic images that it produces. In his days, Wheatstone himself used paper drawings. Today it is more suitable to use digital projection devices. An important aspect is the uniformity in luminance and chromacity of the output device.

Also the flatness of the monitor screens is important. When using heavily curved screens, images will arrive distorted on the mirrors causing poor binocular perception (see section 4.3.2). While

this could in principle be corrected for by distorting the generated images exactly in the opposite direction before they are being sent to the monitor, it is of course easier to use flat monitor screens.
A final quality factor is the frame of the setup. For our experiment the most important is that it is stable.

Binocular colour vision was the crucial aspect of our experiment. Essential for our setup was therefore to provide binocular disparity with equal colours on both screens. To avoid colour calibration problems as much as possible, we used two high quality displays of the same brand and type (*Philips Brilliance 190P* monitors).
In addition to this, the setup consisted of a mirror holder with two right-angled mirrors attached to a frame. To avoid ghosting and chromatic abberation problems, we used front surface mirrors. Input to the monitors was generated by a *800 Mhz Pentium 3* computer with a dual output video card *Asus AGP V9570LE-TD FX5700LE 128MB*. Stimuli were presented in high resolution (1280 x 1024 pixels) to optimise realism as much as possible.

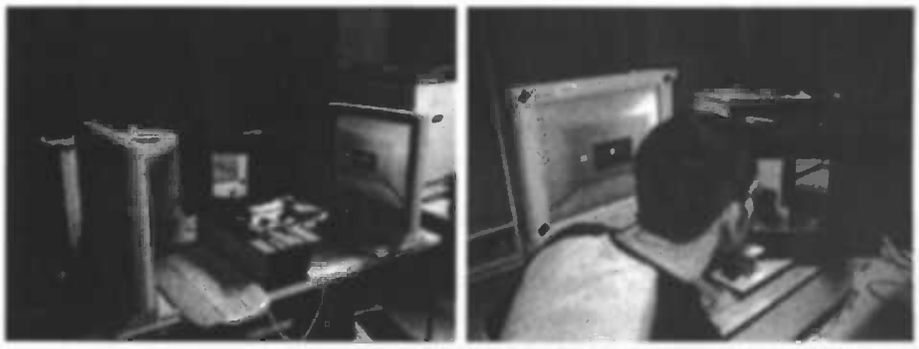
The monitors were facing each other, with the two mirrors placed in between (see Figure 4.1). As a result, after hitting the mirrors, light leaving the monitors will be travelling in two parallel bundles in the same direction. Observers were positioned such that one of these bundles enters the left eye and the other bundle the right eye.
The distance between the monitor and mirror was approximately 45 cm. This distance was chosen such that an average observer would not have accommodation problems. To avoid other light than that from the monitors entering the eyes of the observers, the setup was placed in a room that could easily be made dark.
The first thing that was done after an observer took place in the setup was setting the interocular distance[2] to a value that allowed him to fuse both images.

We should note that the Wheatstone setup is only capable of simulating disparity and convergence; it does not account for accommodation, because the real position of the virtual objects is on flat monitor screens. Therefore, focusing on an object in a particular depth plane does not change the perception in other depth planes, as in real environments. Also no motion parallax cues are provided by the Wheatstone setup. For our experiment this was no problem however, because scenes were static and no head movements were made during the experiments.

4.3 Software

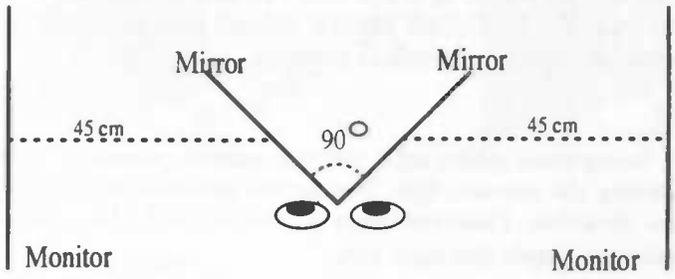
This section describes the essential aspects of the software we created for the experiment. First some background on the Open Graphics Library graphics pipeline is given. Thereafter, we describe the stereo-rendering method and the illumination model used in the experiment.

[2]The interocular distance is a software setting that defines the disparity between both images.



(a) *experimental setup (setup courtesy of Jacob de Jonge)*

(b) *observer behind the setup*



(c) *layout of Wheatstone setup*

Figure 4.1: *Wheatstone setup*

4.3.1 Graphics pipeline

Experiment stimuli were created using the Open Graphics Library (OpenGL). OpenGL is a standard graphics library supported by a range of hardware and software manufacturers. It provides a so-called graphics pipeline that transforms a conceptual model of the 3D world to a 2D raster image. The sequence of steps involved in this process consists of: transformation from model coordinates to normalized device coordinates (NDC); 3-D clipping (if out of bounds); perspective division; determination of colour through lighting equations or depth-cueing; transformation of NDC coordinates to screen coordinates; 2-D clipping (by the screen mask); rasterization (drawing into the frame buffer); and display of frame buffer (see Figure C.1).

Recent developments have made it possible to alter the graphics pipeline. Especially the option to change the pipeline's lighting equations is interesting for us, because it makes it possible to replace the default lighting model with a device-independent CIE based model (implementation of this model will be described in section 4.3.3).

For the discussion of our implementation of the experiment detailed knowledge of OpenGL is not necessary. A more detailed discussion, along with some history of OpenGL, can be found in Appendix C.

4.3.2 Stereo rendering

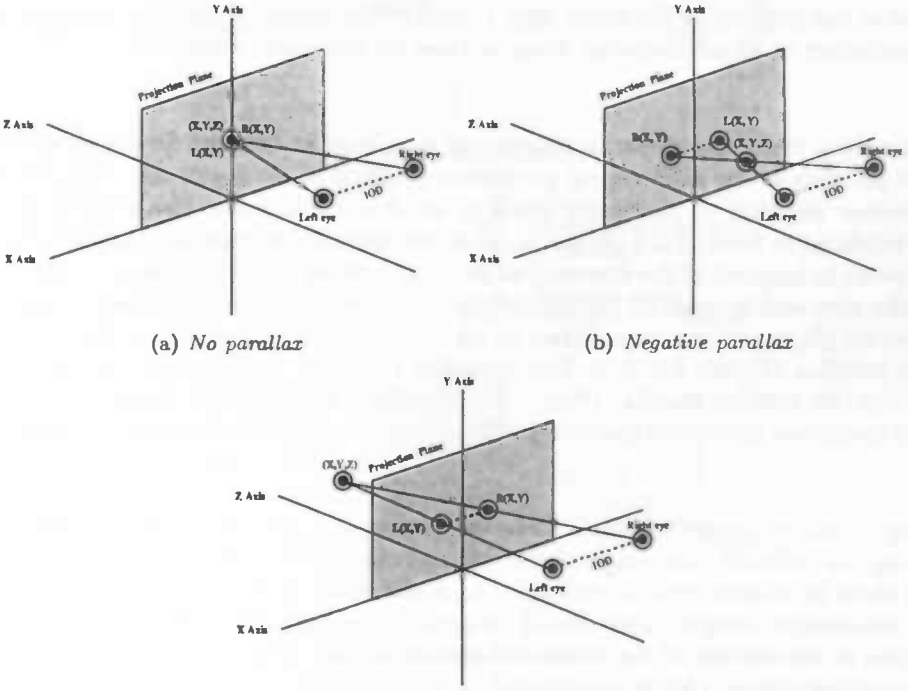
For the experiment that we are setting up, it is important to include depth cues in the stimuli. The most important of these cues is binocular disparity (see section 3.3). In order to establish binocular disparity, we need to present distinct images to both of the observers' eyes in a way such that the visual system is able to fuse them into a single percept. Visual depth information from binocular disparity will then be processed as in normal viewing.
Images presented to the observers in this way also results in correct convergence cues. However, as was pointed out before, accommodation cues are inconsistent because observers are looking at flat images instead of objects that are really in different depth planes. Fortunately, the visual system seems to tolerate this. At least to a certain extent; from literature it is known that as long as the separation of the displays is not more than 1/30th of the distance between observer and display, this inconsistency of accommodation is not noticed by the visual system [2]

By making use of binocular disparity, objects can be placed at different depths planes of the image. Binocular parallax is the difference in projection position of an object onto left and right retina. This difference depends on the depth plane of an object relative to the projection plane. If an object is rendered in front of the projection parallax we call this negative parallax (Figure 4.2 (a)) and it appears to leap out of the screen. The parallax approaches infinity as an object moves closer towards the eyes and is equal to the interocular eye distance when the object is halfway between the projection plane and the eyes. When an object's depth plane coincides with the projection it is at zero parallax (Figure 4.2 (b)). Finally, when an object is positioned behind the projection plane we call this positive parallax (Figure 4.2 (c)) and it appears to be behind (or 'in') the screen. Note that maximum horizontal positive parallax is achieved when an object is rendered at infinity.

Rendering a pair of stereo images is done by placing the virtual camera at different positions (simulating the different viewpoints of the left and right eye). This can be done in various ways. Many of them introduce vertical parallax, i.e., a difference in the vertical height of an object in the two stereoscopic images. This should be avoided, because this will result in the phenomenon that objects at the corners of the image will appear further away from the viewer than objects at the centre of the image. This is experienced as a discomfort.
One of the methods that introduces vertical parallax is the so-called *toe-in* method. Here each camera has a fixed and symmetric aperture and is directed at a single focal point. As can been seen from Figure 4.3 (a). This will result in binocular perception, but the vertical parallax it introduces will cause (increased) discomfort levels. A different way to render stereo images is by using the *off-axis* method (Figure 4.3 (b)). Due to its use of non-symmetric frustums (supported by OpenGL), no vertical parallax is introduced. Since non-symmetric frustums are not available in all rendering packages, the *toe-in* method is still used sometimes.

Whether stereo vision is achieved by the brain depends on the distance from the camera to projection plane (depth planes of very distant objects will coincide like in real vision) and the separation between the left and right camera. Too large a separation can be hard to resolve and is known as hyperstereo. Another issue when creating stereo images is the parallax angle measure, defined as:

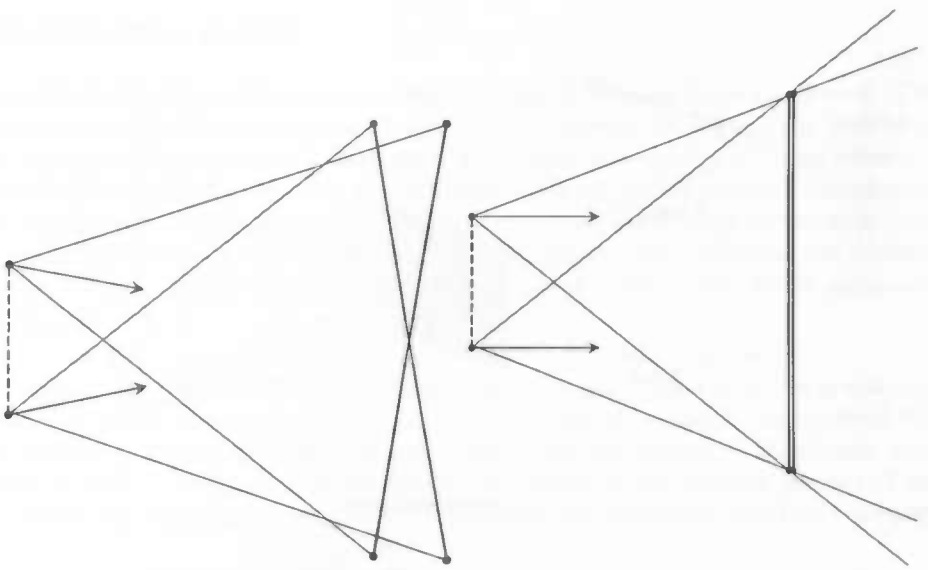
$$\theta = 2\arctan(\frac{Hzp}{2D}) \tag{4.1}$$



(a) *No parallax* (b) *Negative parallax*

(c) *Positive parallax*

Figure 4.2: *In Figure (a), (b) and (c) displays three kinds of horizontal parallax. In Figure (b) objects are perceived as being in front of the graphical display and in Figure (c) objects are viewed as being behind the graphical display. In Figure (a) no depth is perceived*

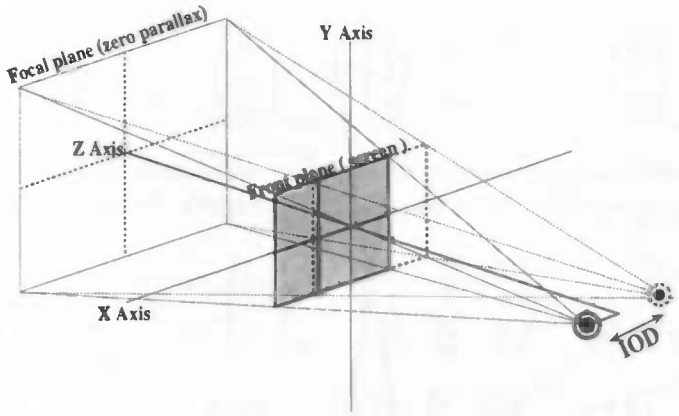


(a) *toe in* (b) *off axis*

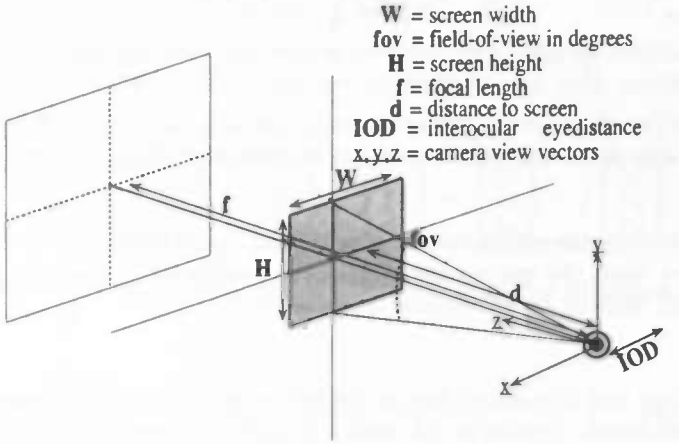
Figure 4.3: *Figure (a) displays the toe in stereo projection, which introduces vertical parallax. In Figure (b) the off axis is shown, which does not introduce vertical parallax*

Were Hzp is the horizontal separation of a projected point between the two eyes and D is the distance of the eye from the projection plane. For comfortable viewing, this angle should not exceed one and half degrees for all objects in the scene.

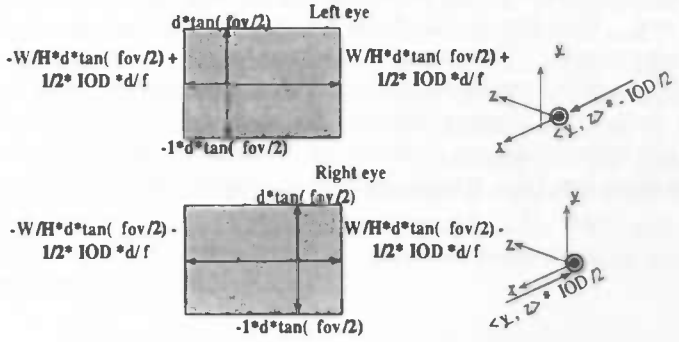
The *Off-axis* method can be used in almost all stereo setups that are present today. We used it for our Wheatstone setup. Details of the used scheme are shown in Figure 4.4.
We begin by placing the virtual camera at the desired position in the virtual scene. The camera determines the viewing distance in the scene and the viewing direction. There are three important parameters to be set. The first is the field-of-view, which is set at an angle that ensures us comfortable binocular viewing. The second is the focal length of the projection plane, which defines the distance at which objects have zero parallax. The last parameter is the interocular eye distance. These parameters define the viewing frustum for each eye, where the offset for each frustum is determined by the interocular eye distance (a distance of zero results in identical frustums). The viewing projections are then transformed to device coordinates for each (calibrated) of the monitors.



(a) *stereo projection*



(b) *frustum parameters*



(c) *left and right frustum + camera settings*

Figure 4.4: *Figure (a) is a schematic drawing of the stereo off-axis frustum. The Figures (b) and (c) display the frustum and camera parameters*

4.3.3 Illumination model

A problem with display devices is that their colourimetric properties differ per device. This means that information about colours in terms of inputs to these devices do not make it possible to determine what the physical properties of the output is. This is undesirable in vision science, because the results of experiments dealing with colour largely depend on the physical stimulations of the eyes of the participants. This problem can be overcome by calibrating the device such that colours can be expressed in terms of a device-independent colour space. We calibrated our display devices (see Appendix A) such that all colours could be expressed in terms of the device-independent CIE 1931 space (section 2.2.1).

Expressing colours in terms of CIE coordinates required a modification of the graphic pipeline's standard lighting model, which uses inputs and equations based on (device-dependent) RGB coordinates. In summary, the requirements for our model were: (i) colours of illuminants and objects are expressed as CIE coordinates, (ii) the lighting equations give a realistic picture of what happens when particular illuminants interact with objects with particular reflectance properties.

In our physical world, the colour of light that is reflected by a surface depends on two factors: illumination properties of the light source and reflectance properties of the surface material. Different combinations of light sources and surfaces can result in the same reflected colour. Hence, the output colour does not allow reconstructing the properties of the light source and illuminated surface.

Usually, illuminant-object interactions are calculated based on an analysis of the different wavelengths of which the incoming light is comprised and the reflectance properties of the object (that is, what wavelengths it absorbs and what wavelengths it reflects). Even though real world surface-light interactions are based on lightwaves, lightwave spectra make colour predictions difficult. Specifying the spectral colour distribution of a colour is too abstract and far removed from human vision. Therefore, we took a different approach. In our model, objects did not have defined absorption and reflectance properties. Instead, the 'colour' of an object was defined by what it looked like when it was illuminated by a (neutral) reference light source, serving as an anchor for object colour. When an object was lighted by a different illuminant compared to the reference illuminant, we looked the shift in colour from the reference illuminant and applied the same shift to the output that it would give with the reference illuminant.

The advantage of this paradigm is that one is no longer constrained to the limited choice of tabulated spectra that are usually used for illuminant-object interaction computations in colour perception experiments. The only restriction is imposed by the boundaries of the colour space bounded by the monitor. Another advantage is that the approach that we used requires less storage and calculations. Finally, this model also avoids the problem of illuminant metamerism [3].

The illuminant-object interactions are being calculated in the following manner[9]. First we transform the CIE (x,y,Y) reference colour to phosphor luminances. The transformation from CIE space to monitor luminance space is achieved by using a conversion matrix, which was determined

[3]Metamerism refers to the situation where two colour samples appear to match under one condition but not under another; the match is said to be conditional. (see also section 2.2)

by the monitor calibration routine. Details on monitor calibration can be found in Appendix A.

$$XYZ \rightarrow Lv: \quad \begin{pmatrix} L_r \\ L_g \\ L_b \end{pmatrix}_{ref} = \begin{pmatrix} \frac{x_r}{y_r} & \frac{x_g}{y_g} & \frac{x_b}{y_b} \\ 1 & 1 & 1 \\ \frac{z_r}{y_r} & \frac{z_g}{y_g} & \frac{z_b}{y_b} \end{pmatrix}^{-1} \times \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{ref} \tag{4.2}$$

This is repeated for the illuminant colour and the surface colour (as under the reference illuminant), which gives us three sets of red, green and blue phosphor luminance values.

$$XYZ \rightarrow Lv: \quad \begin{pmatrix} L_r \\ L_g \\ L_b \end{pmatrix}_{ill} = \begin{pmatrix} \frac{x_r}{y_r} & \frac{x_g}{y_g} & \frac{x_b}{y_b} \\ 1 & 1 & 1 \\ \frac{z_r}{y_r} & \frac{z_g}{y_g} & \frac{z_b}{y_b} \end{pmatrix}^{-1} \times \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{ill} \tag{4.3}$$

$$XYZ \rightarrow Lv \quad \begin{pmatrix} L_r \\ L_g \\ L_b \end{pmatrix}_{srf} = \begin{pmatrix} \frac{x_r}{y_r} & \frac{x_g}{y_g} & \frac{x_b}{y_b} \\ 1 & 1 & 1 \\ \frac{z_r}{y_r} & \frac{z_g}{y_g} & \frac{z_b}{y_b} \end{pmatrix}^{-1} \times \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{srf} \tag{4.4}$$

Then we divide the surface colour luminance and the light colour luminance by the reference luminance. This gives us the surface reflection coefficients and light emission coefficients under the reference illuminant.

$$\begin{pmatrix} \alpha_r \\ \alpha_g \\ \alpha_b \end{pmatrix} = \begin{pmatrix} L_r \\ L_g \\ L_b \end{pmatrix}_{ill} / \begin{pmatrix} L_r \\ L_g \\ L_b \end{pmatrix}_{ref} \quad \begin{pmatrix} \beta_r \\ \beta_g \\ \beta_b \end{pmatrix} = \begin{pmatrix} L_r \\ L_g \\ L_b \end{pmatrix}_{srf} / \begin{pmatrix} L_r \\ L_g \\ L_b \end{pmatrix}_{ref} \tag{4.5}$$

Finally, we multiply these coefficients with the luminance of the reference illuminant to get the simulated reflected light.

$$\begin{pmatrix} L_r \\ L_g \\ L_b \end{pmatrix}_{eye} = \begin{pmatrix} L_r \\ L_g \\ L_b \end{pmatrix}_{ref} \times \begin{pmatrix} \alpha_r & \alpha_g & \alpha_b \end{pmatrix} \times \begin{pmatrix} \beta_r & \beta_g & \beta_b \end{pmatrix} \tag{4.6}$$

The six coefficients are constant for a particular combination of illuminant and surface sample. However, the constants are confounded in the coefficient product α and β, in the same fashion that light and matter are confounded in the wavelength product. The visual stimulus is produced by additive mixture of the three (red, green and blue) phosphor luminances, where each varies in intensity. The reflected phosphor luminance can be transformed back to CIE space if multiply them with the inverse of the conversion matrix. In the program we transform the luminance values to a RGB trichromate, which can be done through a colour look-up function. We used this model to be adapted to the (classic) Phong lighting model (4.1). Although we used this lighting model, the interaction model can be adapted in almost all other lighting models which uses RGB trichromates as (primary) colour indicators.

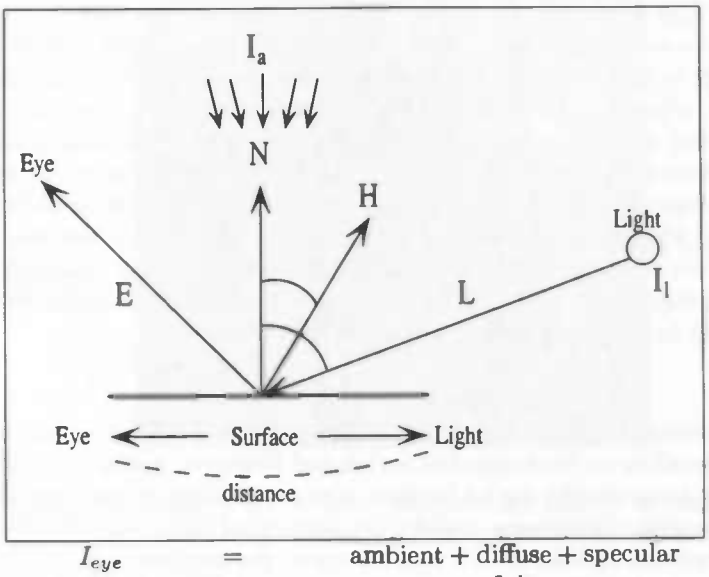
This formula consists of an ambient, diffuse and a specular highlight part. The ambient part represents the colour that is present in a scene but cannot be traced back to a particular light source. This term can be seen as the overall inner reflections and scattering of light throughout the scene. The diffuse part is light that hits a surface and reflects in all directions equal in intensity where this component represents the reflected light that hits the eye. Specular highlight is the part of a light source that is the near or total near reflection of the incident light in a concentrated region around the reflection angle. The intensity of the diffuse and the specular light component is controlled by an attenuation function; as light moves through space the intensity decreases. In the physical world this attenuation factor is the inverse square over the distance, because lights in computer graphics are a representation of reality this term is usually manually set to get the desired result. In the program we dropped the ambient component because we were interested in lighted areas that could be tracked down to a specific light source.

Our model was implemented by altering the graphics pipeline through a vertex and fragment shader. These programs allow us to change the vertex and fragment operations that are present in OpenGL. In the vertex program the angles between a vertex surface normal and the light sources are computed. These angles of incidence present the amount of light that reflects from a surface. The lights in the program are defined in CIE (x, y, Y) space, the incident angle scales the amount of luminance (Y) that is reflected. This gives us for every light a CIE coordinate (x,y) with a scaled luminance (Y) value. All CIE values are then transformed into their tristimulus counterparts (X,Y,Z). Because the tristimulus values are linearly additive we can sum the tristimulus light parts. This tristimulus value is transformed to phosphor luminance [4] by a conversion matrix that is supplied through a pre-performed calibration routine. These luminance values are passed to the fragment shader, where the light-surface calculations and RGB mapping are done as described above.
In the binocular conditions these calculations where done for images of both screens, where each monitor was had its own calibration matrix. The final result is a monitor stereo pair which each display showing the same colours. Although this model is a local illumination model with no shadow abilities, it meets the requirements of our experiment. Global illumination, that produces shadows and interreflecting surfaces, are additive for visual perception but the lack of these visual queues will not result in poor binocular colour perception.

4.4 Experiment

The main goal of the experiment is to test whether binocularity affects our perception of colour (in particular, whether they affect colour constancy and colour induction). To our knowledge, all experiments investigating colour constancy so far used either two-dimensional or three-dimensional stimuli but we do not know of any studies in which dimensionality itself was an experiment variable. In the study presented here, participants carried out a number of tasks that measured their level of colour constancy and induction and we specifically manipulated stimulus dimensionality while keeping all other parameters fixed. If binocularity is a significant aspect of colour constancy or induction, we expect to find different results in monocular and binocular conditions.

[4] Phosphor luminace is the luminance output of a graphical device primary (R,G,B)



I_{eye}	=	ambient + diffuse + specular
ambient	=	$I_a k_a$
diffuse	=	Attenuation $\cdot\ k_d I_l (\boldsymbol{N} \bullet \boldsymbol{L})$
specular	=	Attenuation $\cdot\ k_s I_l (\boldsymbol{N} \bullet \boldsymbol{H})^{\text{(Phong exponent)}}$
Attenuation	=	$\frac{1}{a_0 + a_1 \mathrm{d} + a_2 \mathrm{d}^2}$

I_a	=	ambient light intensity
I_l	=	light source intensity
k_a	=	reflectance properties ambient component
k_d	=	reflectance properties diffuse component
k_s	=	reflectance properties specular component
a_0	=	constant distance coefficient
a_1	=	linear distance coefficient
a_2	=	quadratic distance coefficient
Phong exponent	=	specular component
d	=	distance of travelled light
N	=	surface normal vector
L	=	light source vector
E	=	surface-eye vector
H	=	half-angle vector $\frac{\mathbf{E}+\mathbf{L}}{\lvert\mathbf{E}+\mathbf{L}\rvert}$

Table 4.1: *The Phong illumination model is an illumination model that divides illumination into three components: ambient, diffuse and specular. The ambient term represents the mean spatial illumination that cannot be tracked down to a particular lightsource. The diffuse part is light that hits a surface and reflects in all directions equal in intensity where this component represents the reflected light that hits the eye. Specular highlight is the part of a light source that is the near or total near reflection of the incident light in a concentrated region around the reflection angle.*

4.4.1 Subjects

Nine subjects participated in the experiment, five males and four females (between the ages of twenty and thirty). All participants had normal or corrected-to-normal visual acuity and normal colour and depth vision.
All the participants could be classified as naïve subjects except for one person who is a PhD student with a background in human perception.
Some of the other volunteers participated in an earlier pilot but did not have any knowledge about the questions underlying this experiment.

4.4.2 Stimuli

The stimulus environment that we used consisted of two rooms separated by a window (Figure 4.5). In the experiment, subjects were observers (virtually) standing in the front room facing the room in the back, which was seen through the window. At the back of the front room (next to the window) a target patch was placed which observers were asked to match with the reference patch. The position of the reference patch was an experiment variable and varied across conditions. For all conditions the reference and target patch had the same retinal size. Thus there was no difference in the amount of light that leaves from the patch into the eye.

In a pilot experiment, we noticed that observers tend to consider the colour of the walls in the back room as paint and being illuminated by a neutral light source (instead of being grey and illuminated by a non-neutral light source). Therefore, some additional objects were placed in the back room to provide observers information about the colour and position of the illuminant. The different colours of the objects should aid the subjects' level of colour constancy (constancy fails when a all objects in an environment are mono-chromatic).

Both rooms in the experiment each had its own illuminant (Figure 4.6). The illuminant in the front room was placed in a way such that light from the front room had little or no effect in the back room. The light source in the back room was placed next to the window and was directed at the objects in this room.
Three different illuminants were used: neutral, yellow, and blue. The neutral colour was D65, CIE chromaticity coordinate (0.313, 0.329), which resembles noon daylight. The other colours in the experiment were (0.380, 0.376) (yellow) and (0.279,0.292) (blue). These conditions resemble sunlight during the evening and morning, respectively.
The chromaticity of the illumination in the back room varied across conditions while the front room was always illuminated by the neutral illuminant.
The light intensity in the rooms could vary from zero (no light) to forty candelas. The exact value depended on the distance between an object and the light source and the angle of incidence. The overall intensity was between twenty and thirty candelas.

The binocular properties were set in a way such that there was no negative parallax in the stereo images. (This was done because negative parallax can is a strain on the observer at close distance viewing.) Before starting the experiment, the interocular eye distance of the program was set such to that the subject was able to fuse both images into one percept.

Figure 4.5: *Stimuli presented to the subject. Subjects his task is to match the chromaticity and luminance of the target patch to the reference patch, where the reference patch is viewed on a per condition changing spatial surround.*

The textures that were used in the stimuli only affect the amount of luminance reflected by a surface. This gives a higher degree of realism, but does not influence the chromatic properties of an object. The surface properties are stored in the vertices of an object and colour interactions are done per vertex. The program is created such that it can run on one or two screens, where each screen has its own calibration. When running in a stereo setup, camera view points and frustums are set such that perfect binocular image can be produced.

4.4.3 Task

The task of the subjects was to match the colour of the target to the reference patch (such that they appeared equal). The luminance and chromaticity of the target patch could be altered by using the mouse. Moving the mouse while holding the right mouse button altered the chromaticity coordinates of the patch and using the scroll button changed the luminance. For this experiment we used the CIE 1931 colour space.
The target and reference patches were not affected by the light sources (these patches were directly copied to the pixel buffer). The colour and luminance of the reference patch depended on the condition (see below); the colour and luminance of the target patch were set to a random CIE coordinate in each condition.
To avoid after effects (due the getting adapted to a particular stimulus colour), there was a blank interval of ten seconds between each two consecutive trials. A subject could take little breaks between trials, because no recordings were done in between trials. Before a subject began his or her first session the task was explained. In addition, the subject was familiarised with the colour space that would be used in the experiment.

The reference and target patch were equiluminant. Nonetheless, there could be perceptual differences due to differences in the backgrounds of the patches. Hence, changing the luminance would

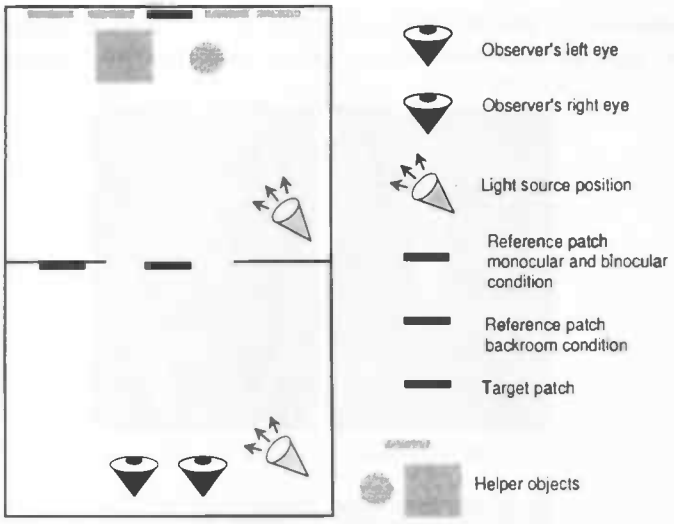


Figure 4.6: *A subject is an observer standing in the front room looking into the back room. The experiment contains three conditions. The target patch is placed onto the left wall in the front room for all conditions. The position of the reference patch varies in each condition. In the colour constancy condition the patch is placed onto the back room wall and in the colour induction conditions the patch is placed in the middle of the window.*

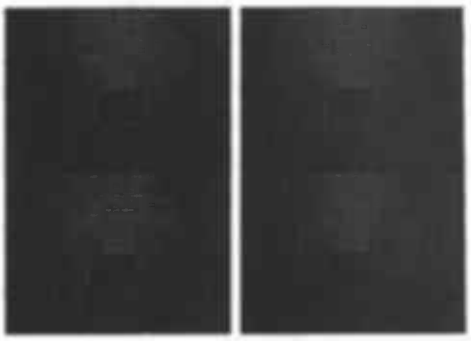
aid subjects in their task and we therefore provided with the ability to alter luminance as well, although our primary interest concerned their chromaticity settings.
After setting the interocular eye distance, subjects performed a short training session in which he or she could get accustomed to the dark environment and be familiarised with making settings in CIE space. If the subject was confided with his or her colour setting skills and was accustomed to the dark, the actual experiment was started.

4.4.4 Conditions

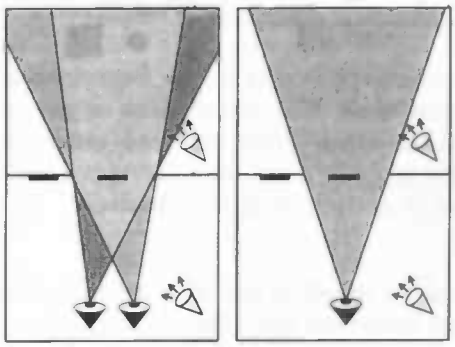
The reference patch could be placed at two different locations: against the wall of the back room (we will refer to this condition as 'back room') or at the centre of the window that separates the rooms. For the latter position, the back room was perceived either monocular (we will refer to this condition as 'monocular') or binocular (we will refer to this condition as 'binocular').
For the monocular conditions, we used the stencil buffer to cut out the window. In addition, in this condition the front room was rendered with the camera distance set to the subjects' interocular eye distance, while the back room was rendered with both cameras at the same position (Figure 4.7). This ensured us that the front room would keep its binocular properties while the back room was seen in a monocular state. This resulted in the same depth cues in both conditions with exception of the binocular one.

All conditions were performed with three different illuminants in the back room: neutral, yellow, or blue (with CIE coordinates as described in section 4.4.2). The neutral illuminant was necessary as a reference point for the shifts being made under the other illuminants.

The experiment consisted of one hundred and eight trials (3 conditions x 3 illuminants x 12 repetitions) and was split up into two sessions. Conditions were presented in random order.



(a) *Binocular view* (b) *Monocular view*



(c) *Binocular and monocular viewpoint*

Figure 4.7: *In Figure (a) the back room is viewed with a binocular disparity angle, while in Figure (b) the back room is rendered with no disparity angle. Figure (c) is a schematic drawing of both conditions.*

4.4.5 Analysis

In the 'back room' condition the reference patch was placed at the back wall and appeared to be illuminated by the back room illuminant. If a subject possesses perfect colour constancy, he would perfectly compensate for the illumination and perceive the reference patch as grey. With no constancy at all, however, the subject would not compensate at all and give the target patch the same colour as that of the illuminant in the back room.
The data from the back room condition allowed us to determine whether our virtual environment induces some degree of colour constancy.

The data from the other two conditions were used to test whether binocularity affects colour induction. In these conditions the target was placed in the centre of the window and we investigated whether there is a difference in perception by looking at the (possible) chromatic induction effect in the binocular and monocular conditions.

The reference patch in these conditions had an achromatic colour (D65). Colour induction is the influence of surrounding (or background) colours to an observed patch.
Because of the non-uniformity of the CIE 1931 space we transformed our data to CIE Lab space prior to the analyses.

Chapter 5

Experimental results

In this chapter we discuss the results of the conducted experiment. In the "back room" condition we investigate whether the setup with the presented stimuli induces colour constancy. Here the observers matched the target to the reference patch, where each patch was in a different room. The "monocular" and "binocular" condition determined whether differences in binocular and monocular perception of the spatial surround influences the level of colour induction.

5.1 Transforming to *CIE Lab* space

Because of the non-uniformity of the *CIE 1931* space we transform our data to *CIE Lab* space. The transformation gives us three axes, where the a axis is the change from blue (negative) to yellow (positive), the b axis is the change form green (negative) to red (positive), and L represents the amount of luminance (brightness). In the different conditions, we examine only the chromatic changes. Therefore we drop the L axis. The transformation involves a reference colour which represents the origin of the colour space. In Figure 5.1 the settings for all subjects are transformed to *CIE Lab* space, while Figure 5.2 the settings of subject AK and ME as examples of individual subject data. In Figure 5.1 (a) we see a large difference between the settings for each illuminant condition. Under the yellow and blue illuminants, the settings are shifted toward the colour of the illuminants (backroom condition) relative to settings under the neutral illuminant. This suggests that subjects have low colour constancy. In Figure 5.1 (b) and (c) we also see differences between the settings for each illuminant condition. Here the settings under the blue and yellow illuminants are shifted in the opposite direction to the illumination, which implies a colour induction effect.

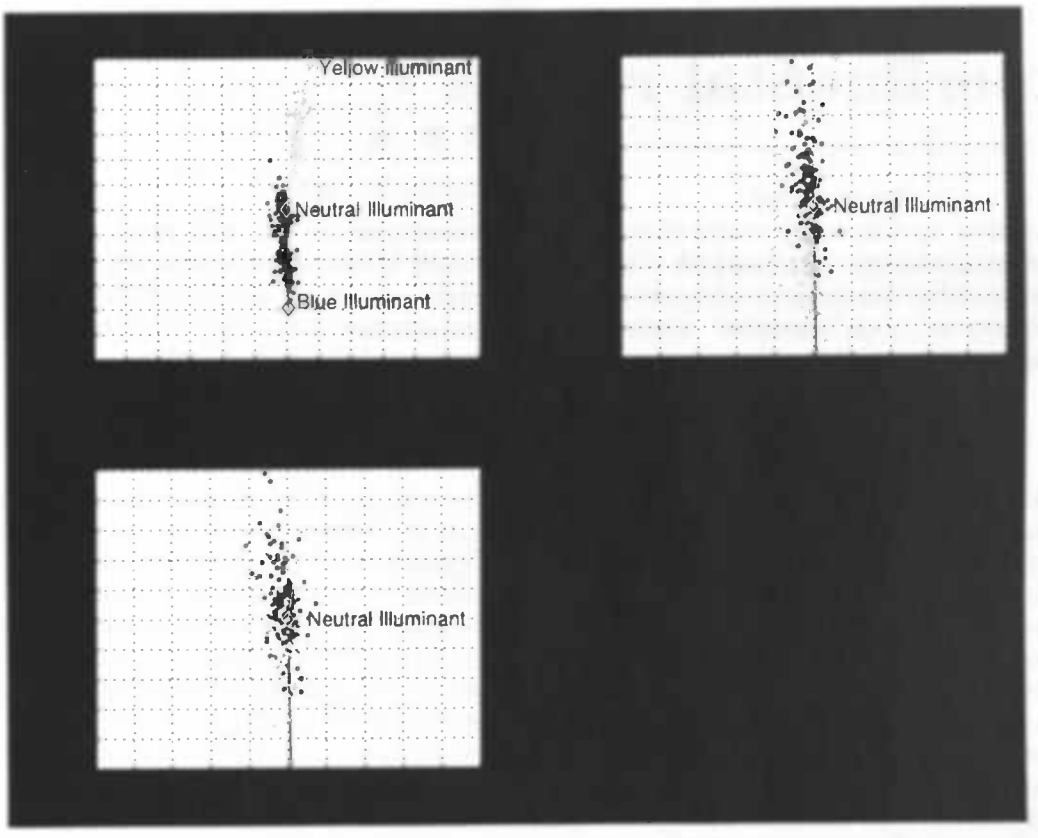
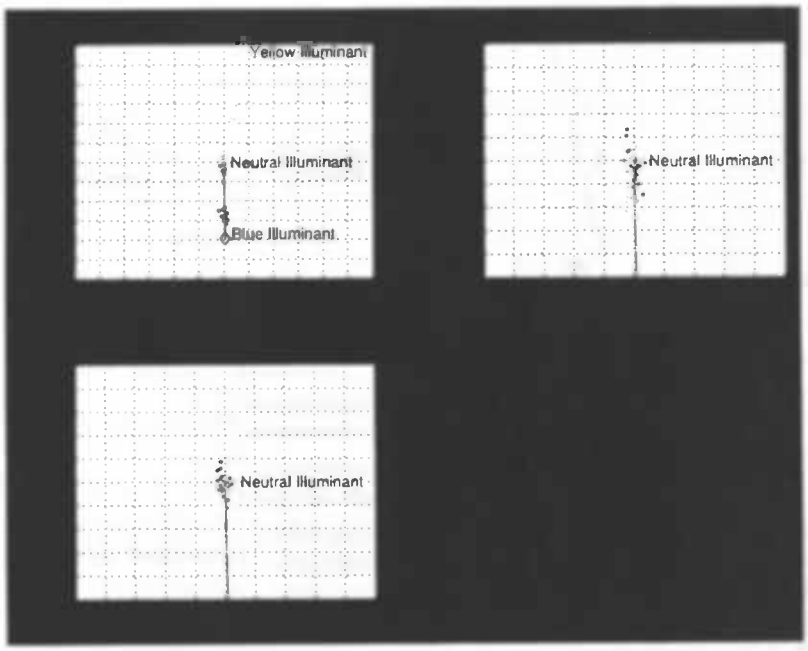
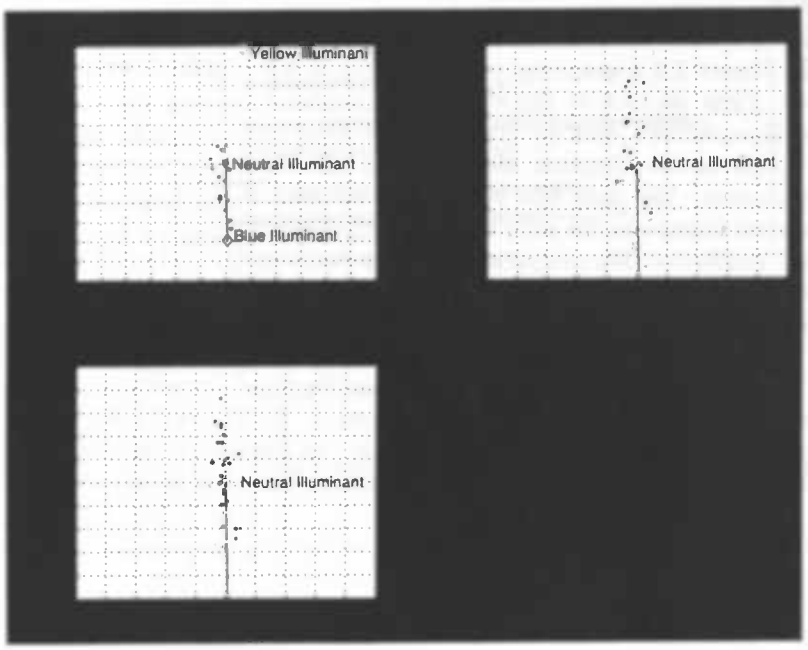


Figure 5.1: *Test results of the subjects under different conditions in CIE Lab space. Each subject had to match a target to a reference patch. Figure (a) shows the results under the colour constancy condition and figures (b) and (c) display the effects of colour induction under the monocular and binocular conditions, respectively. Each dot represents an individual trial. The colours of the dot represent the colour of the illuminant in the back room.*



(a) AK



(b) ME

Figure 5.2: *Colour settings made by subjects AK and ME in CIE Lab space. (Description see Figure 5.1)*

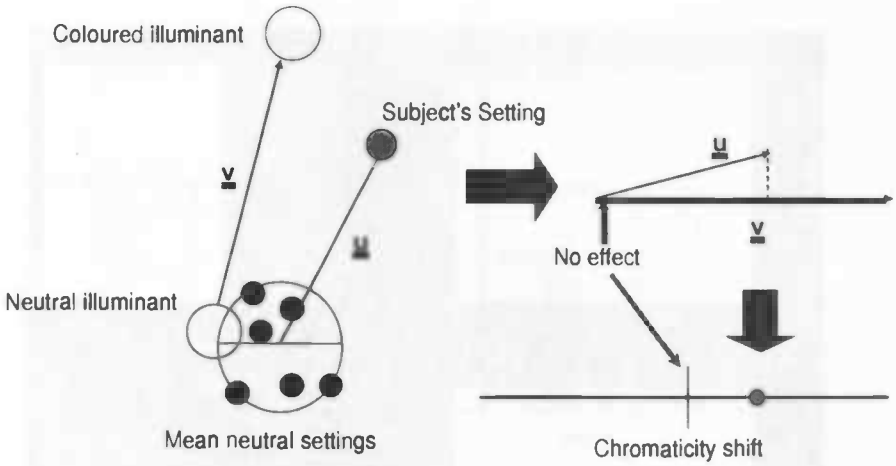


Figure 5.3: *Vector v gives us the direction and length of the chromatic shift between the neutral and coloured illuminants. Vector u is computed from the subject's mean setting under the neutral illuminant relative made to the setting under that illuminant. We project the vector u onto the vector v. This gives the one-dimensional projection of the colour shift associated with the illuminant.*

5.1.1 Inspection of the illuminant effect

In the following section we quantify the differences between the settings under each illuminant condition. In Figure 5.3 a graphical representation is shown. We do this transformation in the following way. First we look at the shift in chromaticity between the neutral and the coloured (blue,yellow) illuminants. Here we draw a vector from the neutral to each coloured illuminant. This vector gives us the direction and length of the chromatic shift between the neutral and coloured illuminants. We term this vector u. The second step is to compute the mean setting under the neutral illuminant for each subject. This tells us what a subject perceives as a neutral colour in that condition. We compute the vector between the subject's mean setting under the neutral illuminant and the settings made under the blue and yellow illuminants. We term this vector v. The next step is to project each of the vectors associated with subjects' settings u onto vector v. This gives the one-dimensional projection of the colour shift associated with each illuminant. Our statistical analyses are performed on these projections. In Figure 5.4 the shift for every subject, illuminant and condition is plotted. In Figure 5.5 the data are pooled across subjects but within each condition and illuminant. In Figure 5.6 the data from each illuminant is also pooled to give an overall comparison between the back room, monocular and binocular conditions.

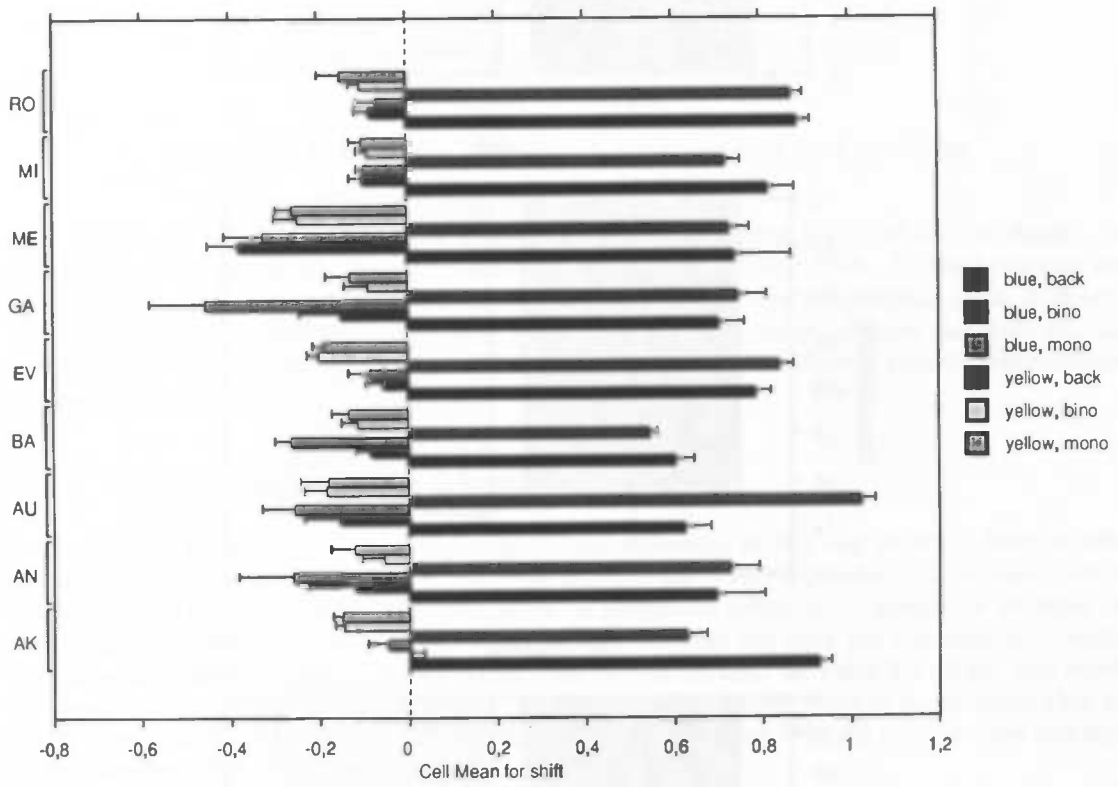
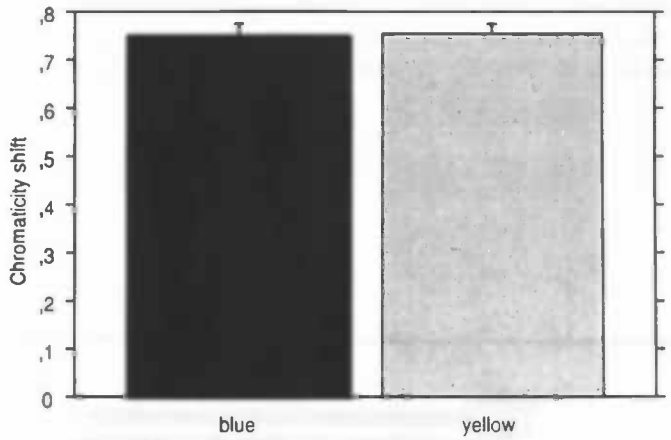
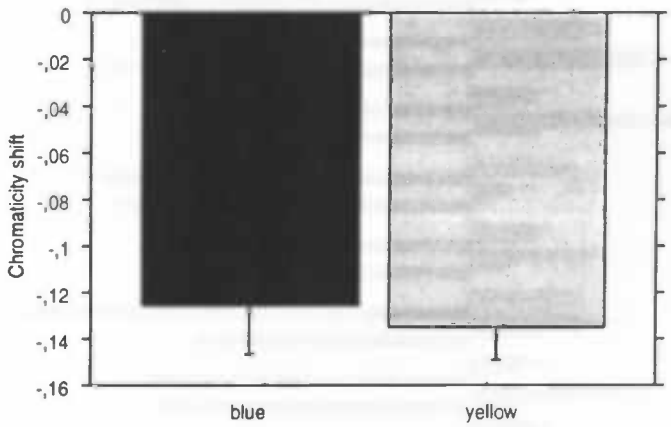


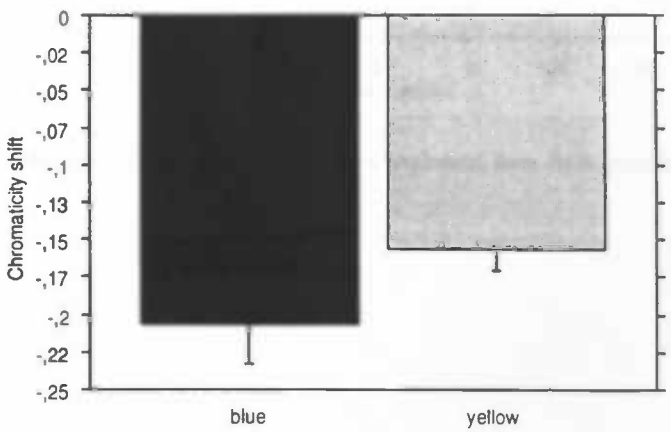
Figure 5.4: *Mean shift and standard error for each subject, illuminant and condition.*



(a) *Back room*



(b) *Binocular*



(c) *Monocular*

Figure 5.5: *Mean shift and standard error for each illuminant per condition.*

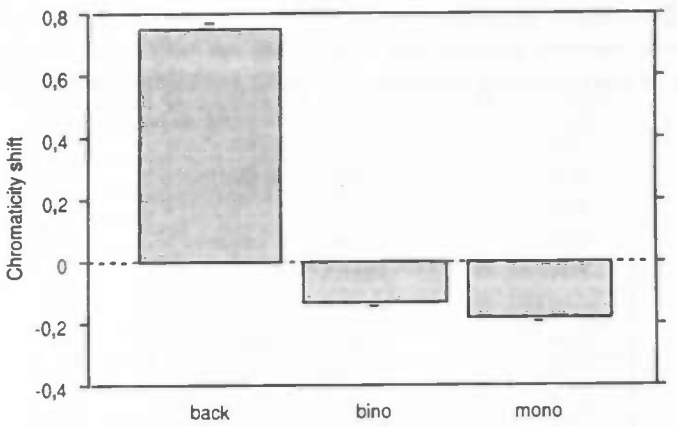


Figure 5.6: *Mean overall shift and standard error for each condition.*

Figures 5.5 and 5.6 show that colour constancy is generally very poor. More specifically, the shift in perceived colour is almost as large as the total illumination shift. In the binocular and monocular conditions we see a shift in a direction opposite to the illumination shift, indicating that there is an induction effect. At a first glance we see a difference between the binocular and monocular conditions. In the following section we examine whether there is a statistical difference between the two conditions.

5.2 Binocular versus Monocular

In this section we examine if there is a significant difference in the size of the induction effect with and without binocular cues. We therefore perform an *ANOVA* (analysis of variance) test on our data. The results from the *ANOVA* gives us statistical evidence to determine whether the conditions are similar or different. An *ANOVA* assumes that the data are normally distributed. Therefore we perform a Kolmogorov-Smirnov test on the dataset. In Table 5.1 all the test results are shown for each subject, condition and illuminant. If H=0 then there is no evidence that the distribution is different from normal with significance level set at 5%. All the P-values displayed are greater than this level of probability.
From Table 5.1 we can conclude there is no statistical ground to reject the hypothesis of normally distributed data. In the *ANOVA* we use the shifts as our dependent variable and the *subject*, *illuminant* and *condition* as independent factors.
In Table 5.2 the results of the *ANOVA* are shown. In the first column we can see the different factors and interaction effects between factors. We are most interested in the P-values of the different factors and interaction terms. The factor we were most interested in was the difference between the monocular and binocular condition. From the table we can conclude there is a statistically significant difference between these two conditions ($p<0.01$). This strongly indicates that binocular disparity has an influence on the amount of colour induction.
The second important aspect of the table is the low P-value associated with the factor *subjects*. This strongly indicates that the subjects all have signicantly different reactions to our presented stimuli ($p<0.0001$).
The third factor examines the effect of the two illuminants. Is the average shift between the

blue and yellow illuminants significantly different? The answer is no (p=0.31). If we look at the interaction term between subject and illuminant we notice the P-value is again very small ($p<0.001$), telling us that subjects respond differently to different illuminants.

subject	condition	illuminant	H	P-value
AK	monocular	blue	0	0.839
AK	monocular	yellow	0	0.406
AK	binocular	blue	0	0.968
AK	binocular	yellow	0	0.636
AN	monocular	blue	0	0.204
AN	monocular	yellow	0	0.315
AN	binocular	blue	0	0.448
AN	binocular	yellow	0	0.436
AU	monocular	blue	0	0.614
AU	monocular	yellow	0	0.389
AU	binocular	blue	0	0.622
AU	binocular	yellow	0	0.768
BA	monocular	blue	0	0.827
BA	monocular	yellow	0	0.397
BA	binocular	blue	0	0.613
BA	binocular	yellow	0	0.936
GA	monocular	blue	0	0.176
GA	monocular	yellow	0	0.782
GA	binocular	blue	0	0.355
GA	binocular	yellow	0	0.729
EV	monocular	blue	0	0.146
EV	monocular	yellow	0	0.217
EV	binocular	blue	0	0.552
EV	binocular	yellow	0	0.526
ME	monocular	blue	0	0.096
ME	monocular	yellow	0	0.401
ME	binocular	blue	0	0.261
ME	binocular	yellow	0	0.661
MI	monocular	blue	0	0.100
MI	monocular	yellow	0	0.411
MI	binocular	blue	0	0.305
MI	binocular	yellow	0	0.453
RO	monocular	blue	0	0.154
RO	monocular	yellow	0	0.630
RO	binocular	blue	0	0.470
RO	binocular	yellow	0	0.623

Table 5.1: *Kolmogorov-Smirnov goodness-of-fit normality hypothesis test on the subjects' colour induction chromaticity shifts under both illuminants. If H=0 then there is a probability with significance level above 5% that the data are normally distributed.*

	DF	Sum of Squares	Mean Square	F-Value	P-Value	Lambda	Power
subject	8	1,809	0,226	5,689	<0,0001	45,510	1,000
illuminant	1	0,040	0,040	1,010	0,3156	1,010	0,162
condition	1	0,277	0,277	6,979	0,0086	6,979	0,759
subject*illuminant	8	1,118	0,140	3,514	0,0006	28,114	0,986
subject*condition	8	0,371	0,046	1,167	0,3181	9,333	0,537
illuminant*condition	1	0,095	0,095	2,391	0,1229	2,319	0,321
subject*condition*illuminant	8	0,263	0,033	0,826	0,5799	6,609	0,381
residual	396	15,741	0,040				

Table 5.2: *The table shows the results of an ANOVA performed on the experimental results. We find significant effects of the factors, subject, condition, and the interaction subject*illuminant.*

Chapter 6

Conclusions

This dissertation examined colour perception in a virtual scene. We were particularly interested in the questions whether (a) the virtual scene that we developed produces colour constancy and (b) the amount of depth information affects the amount of colour induction.
The amount of colour constancy was similar to that which has been found in previous studies with flat two-dimensional displays [1]. This result is may be surprising because experiments with real three-dimensional displays sometimes get much higher levels of constancy, up to 80% [5].
A possible explanation why we do not find such high levels of constancy is that the illumination used in our virtual scene model was not realistic enough. It can be that other colour cues are needed to be included the scene as well (e.g., highlights and shadows). One way to improve the scene would be to use a global illumination model, rather than the local model used in the current study.

Another factor that exists in real scenes but not in virtual scenes is accommodation to objects that are in different depth planes. Absence of this important monocular depth cue might be another factor reducing the realism (and, thereby, the level of colour constancy) in our scene.
The results also show that depth information caused by binocular disparity affected the colour induction effect. The amount of colour induction was greatest when the target and inducing regions were in the same depth plane. Our results strongly indicate that binocular disparity can modulate colour perception, something that was also found in previous studies. (Shevell and Wei, 2000)[11], for example, showed that the greatest effect of a remote checkerboard region on a target and its surround occurred when all stimuli were in the same depth plane.
Our results show directly that induction in a relatively complex virtual scene is greatest when all stimuli lie in the same depth plane. The induction effect that we found could be larger if the spatial surround consisted of more saturated colours (i.e., when using less natural illumination).
In conclusion, our study shows that virtual scenes can successfully be used as a tool for examining aspects of the interactions between colour and depth perception.

Appendix A

Monitor Calibration

Visual experiments that use computers to generate virtual environments need to be certain that a colour that is computed also is exactly represented. Calibrating is an essential part in vision experiments that make use of any projecting device to show different stimuli. Results can only be justified if the stimuli used in vision experiments display the colour we defined. The concern here is the correct mapping from human perceptual space to a computer generated display presentation. While human perception can be seen as a unique device- independent colour space, based on matching functions over humans perceptual interval of wavelengths. Computer rendered images are device dependent. The problem here lies in the graphical output device (and sometimes even the underlying operating system). Computers map a colour in RGB space to a certain channel output. A problem here is the great difference in channel primaries and output among graphical output devices.

Almost all monitors that are on the market today, (CRT's and TFT's) make use of mixing three primary colours (red, green and blue) to produce a palette of different colours. One problem concerning the mixtures of colours that also arises is the non-linearity between the computers RGB space and monitor output. This problem becomes avid when displaying a certain RGB mixture over a range of different intensities. At low RGB ranges there is almost no increase of brightness visible, while at higher levels there can be jumps in brightness. A different aspect is the shift in colour, because the mixtures are not stable for different brightness values.

Another drawback is the range of colours a monitor can produce, while human beings can see a widely range of different colours, today's graphical output systems only can reproduce a small area of these colours. Picking the device to visualise experiments plays an important aspect in the field of perceptual research. Not only the colour representation, but screen size and refresh rate for instance are some of the other issue's that are a factor in perceptual experiments. But these are issues beyond the scope of this chapter.

There are some perquisites a monitor must have; Monitors must have independent colour channels, no channel interaction must exist. Colours are created from mixing the different channel outputs, the luminance of the represented colour should be the same as the summed luminance values. These means no after sampling/effects should take place or the device has a fourth brightness gun, (which is sometimes present in beamers) Gun primaries must remain stable after warming up. The colour on the screen should be equilluminant, colours are not to be different looking from a corner or the middle part a screen. While some of these problems can be overcome, without these constraints predicting colours becomes a mind boggling mathematical and computational complex problem.

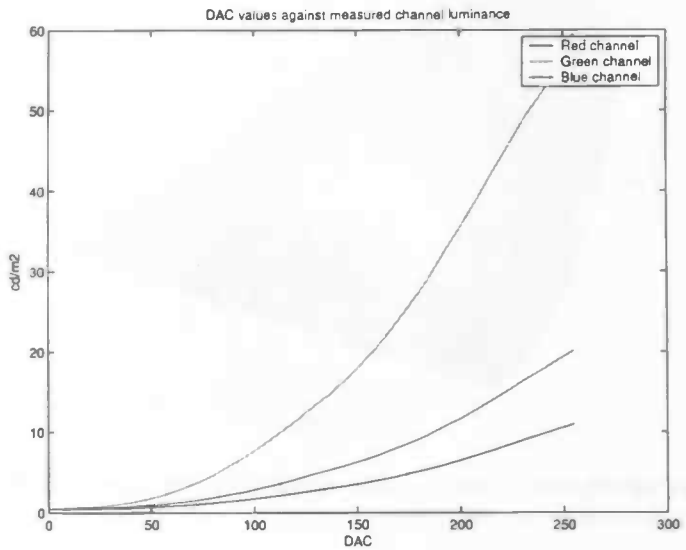
Before the calibration, let the monitors warm up for about 20 minutes for TFT's and an hour for CRT screens. This ensures getting a stable measurement. When the monitors are warmed up, we position the *Minolta CS-100a* in front of the screen. The Minolta is a chromatic meter that measures an illuminant or reflection of an illuminated object that we aim at. The light that hits its lens is measured and expressed in CIE space. Then we start the calibration program. (The program and the Minolta are explained in appendix B). We let the program measure a range of DAC values. DAC are digital values that are transformed to analogue signals by the monitor. The term DAC is an alternative for the term RGB, although they are different in definition, they have the same meaning. They represent the output of a colour channel. On today's graphics hardware, colours are represented as 24 bit values, where every channel primary is eight bits. This gives every monitor channel 256 discrete values. A lower number of bits can be used, but this gives a loss in precision. A calibration routine usually consists of measuring each gun range from 0 to 255 in a predefined number of steps. The step size is usually between four and eight; the size usually depends on how good we can fit a polynomial to the data.

General monitor settings are; putting the brightness level down to a certain range of experimental interest and setting the contrast level at its highest. Lowering the brightness decreases the overall luminance of a monitor. Perception experiments usually take place in a dark room. This means that we often are satisfied with luminance values that are lower than a monitor can produce. Lowering the luminance setting to the range we need gives us a higher discrete interval over the luminance range; therefore increasing the precession a colour can be made. Setting the contrast at its highest ensures no channel interaction. Some models may induce channel interaction if the contrast is not at its highest level. These are not standard setting, but come from a history of experience dealing with monitors. If some uncertainties do arise with the monitor that is used, consult the manual or the manufacturer. Below the data measured by the Minolta is given for one of the monitors used in the setup.
From Figure A.1 the CIE x,y values and luminance output for each colour channel can be read. The exists some irregularity in CIE x,y and luminance values at lower DAC values. This does not mean colours at low luminance values are different than colours at high levels, but indicate the sensitivity of the Minolta and the behaviour of monitors. From the graphs it can be seen that when the luminance is above a certain level the chromaticity of a colour channel begins to stabilize. Because all monitors channel output behave in an exponential way, the luminance range at the lower DAC values is always smaller than the range covered by the higher values. From an experimental point of view we can always drop the lower DAC values, because the luminance levels are to low for good colour perception. A topic about tristimulus values and the CIE colour space are discussed in a different chapter.

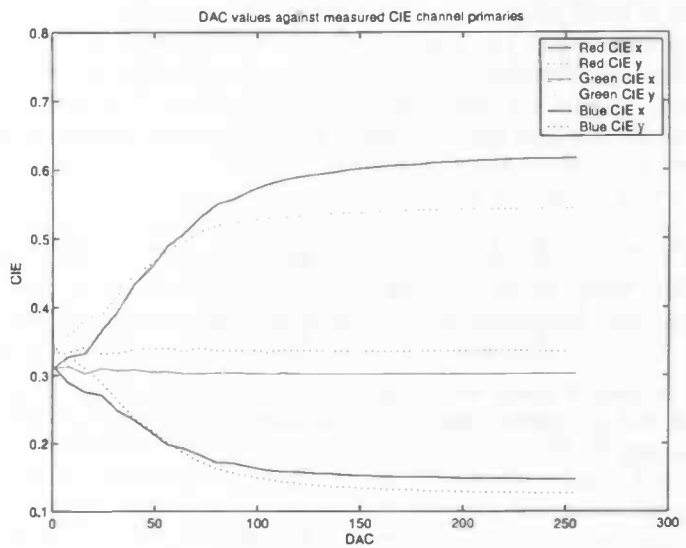
Transforming a colour from CIE (x,y,Y) space to RGB space is done first by transforming the CIE value to the (X,Y,Z) tristimulus counterpart and change these to the appropriate luminance value for each gun. These luminance levels have to be mapped back to the corresponding RGB trichromate. A CIE (x,y,Y) colour can be seen as a vector in CIE space. Transforming a (x,y,Y) value to a tristimulus value is done by the following standard formula.

$$\begin{aligned} X &= \frac{x}{y} \cdot Y \\ Y &= Y \\ Z &= \frac{(1-x-y)}{y} \cdot Y \end{aligned} \tag{A.1}$$

Changing the tristimulus values to luminance values need to be converted through a transformation



(a) *Luminance values*



(b) *CIE values*

Figure A.1: *Figure (a) shows the measured luminance for each colour channel. In Figure (b) the measured chromaticity coordinate is shown.*

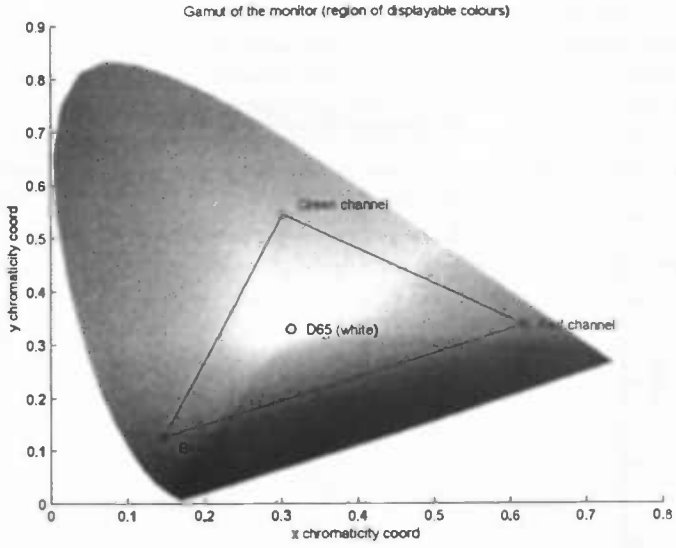


Figure A.2: *Display of the area of colours this monitor can produce in CIE 1931 space (gamut). The boundaries are formed by the red, green and blue channel primaries.*

matrix. The matrix is build upon the monitor primaries from each gun. Every primary produces a different CIE x,y coordinate, the region enclosed by these colours is called the gamut. All the colours a monitor can produce fall in this region, colours chosen out of this region will not be displayed properly. Again we change these values to (X,Y,Z) values. Tristimulus values are additive, this means we can sum the tristimulus values for each channel to get the colour on the display.

$$\begin{array}{rcccccc} X & = & \frac{x_r}{y_r} \cdot Y_r & + & \frac{x_g}{y_g} \cdot Y_g & + & \frac{x_b}{y_b} \cdot Y_b \\ Y & = & Y_r & + & Y_g & + & Y_b \\ Z & = & \frac{(1-x_r-y_r)}{y_r} \cdot Y_r & + & \frac{(1-x_g-y_g)}{y_g} \cdot Y_g & + & \frac{(1-x_b-y_b)}{y_b} \cdot Y_b \end{array} \tag{A.2}$$

We can put these in matrix form, we change the Y_r, Y_g and Y_b to Lv_r, Lv_r and Lv_g to make a distinction between the generated luminance from each channel and the tristimulus value that is produced on the screen.

$$z = (1 - x - y)$$

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} \frac{x_r}{y_r} & \frac{x_g}{y_g} & \frac{x_b}{y_b} \\ 1 & 1 & 1 \\ \frac{z_r}{y_r} & \frac{z_g}{y_g} & \frac{z_b}{y_b} \end{pmatrix} \times \begin{pmatrix} Lv_r \\ Lv_g \\ Lv_b \end{pmatrix} \tag{A.3}$$

If we want to transform a tristimulus value back to luminance values, we need to invert this formula. We do this by inverting the matrix. Inverting the matrix is always possible because each gun primary can be seen as a unique independent basis vector in CIE space, therefore the matrix

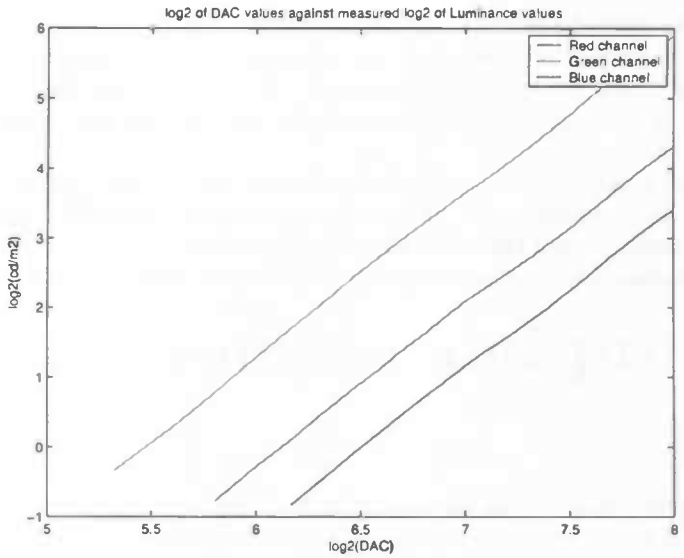


Figure A.3: *In this figure the log_2 luminance values are shown for each colour channel.*

is never singular.

$$\begin{pmatrix} Lv_r \\ Lv_g \\ Lv_b \end{pmatrix} = \begin{pmatrix} \frac{x_r}{y_r} & \frac{x_g}{y_g} & \frac{x_b}{y_b} \\ 1 & 1 & 1 \\ \frac{z_r}{y_r} & \frac{z_g}{y_g} & \frac{z_b}{y_b} \end{pmatrix}^{-1} \times \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{A.4}$$

The luminance values for each gun have to be mapped back to RGB values. Here we need the luminance values measured from each primary. There are some constraints these luminance values must have; values must be monotone continuous increasing and abide a one to one relationship. If these constraints can not be satisfied, a modification is needed. Sometimes there is an irregular interval or the output does not behave like an increasing function. Solutions are; fitting a smooth increasing function to the luminance data or disregard the areas where the constraints are not meted. Although these changes ensure us a proper luminance range, manipulating data points is being paid in loss of colours that can be represented or precession.

There are many different ways how to map the luminance values to the appropriate RGB index values. A simple solution would be to make a look-up table for every primary. While a look-up table is simple to understand it is difficult to create. Creating a look-up table is knowing every RGB-luminance pair. A head on approach is measuring the whole range for every primary, while this ensures us of a complete look-up table, measuring is a time-consuming operation. A better solution is to fit a function to the data. Before we fit any function or method to the data, we need to correct for the dark current. The dark current is the ambient light which is present even though no colour is produced on the screen. Dark current correction is done by subtracting the zero DAC value measurement from all the other data points. After this we take the log of the points, transforming the data to which a function can be better fitted.
Here we fit a polynomial to the data. We select from these data points a regular interval, usually this means dropping the first measurements. These often fall out of order with the other data points. A third degree polynomial will often do the trick, if the error is lower than halve a DAC

value all the points from the created polynomial will fit tightly to the data. Sometimes the fit is awful at lower DAC values; fortunately these are always at low luminance levels, which will not be used in typical colour perception experiments. The points from the fitted function must be transformed back to the original data. Inverting the log function transforms the fitted points to the original DAC range.
Instead of using a look-up table, the polynomial we created here can be used. Even though calculating polynomial points comes at a higher price than looking up values from a table. A function is easier implemented than creating a look-up table.
Now the whole calibration pipeline is completed, we can implement this into our experiment program.

Appendix B

Calibration routine and program

In the following section we will discuss the calibration routine and program used to get a calibrated monitor. The routine makes use of the Minolta chromameter cs-100a and a personal computer with a Windows operating system. The Windows platform is needed to run the program. All the measurements are done by the Minolta. The program displays the different calibration stimuli on the screen and handles the communication with the Minolta. Before we talk about the program we will talk about the Minolta and some necessary steps needed to be done before starting a calibration.

The *Minolta CS-100a* is a lightweight, compact handheld meter for taking non-contact colour measurements of light sources or reflective surfaces. By the trough-the-lens viewing system we can accurately indicate the area to be measured. The CS-100a has one degree acceptance angle and reduces flare which results in measurements that are unaffected by source outside the area. The CS-100a has three high sensitive silicon photocells that are filtered to closely match the CIE (1931 second degree) Standard Observer Response. Signals from the sensors are processed by a built-in microprocessor. Luminance data is shown inside the viewfinder, outside on the external display on the side of the unit both luminance and chromaticity are shown. Measured data has a two percent error in luminance and a 0.004 error in chromaticity. The CS-100a is equipped with a data-output terminal for sending data to a separate computer. Before we can use the Minolta for a calibration routine we need to make sure all the settings of the meter are correct. Luminance unit selector tells us in which quantity we want to express the measured luminance, to work with our program we set this to $\frac{cd}{m^2}$. We set the measuring mode selector to ABS, if we put the selector to DIFF the difference in luminance and chromaticity is computed. Calibration selector switch is set to PRESET; this gives us measurement data that match the CIE matching functions. The Response speed selector switch is set to SLOW, because we are doing measurements in a dark-room with a flickering monitor, here we need more time to perform a stable measurement for each stimuli presented. The selector switch can be set to FAST if we take measurements done under natural light. Later we will discuss the communication between the Minolta and the program.

First of all, the monitor that is going to be calibrated should be placed in a dark environment and needs to warm up. Warming up the monitor is essential for CRT monitor, because their gun output can change drastically if not proper warmed up. Make sure a CRT monitor warms up for at least an hour; this will ensure no large changes in gun output. TFT monitors usually do not need a long warm up period; a warm up period of twenty minutes will give stable readings. When

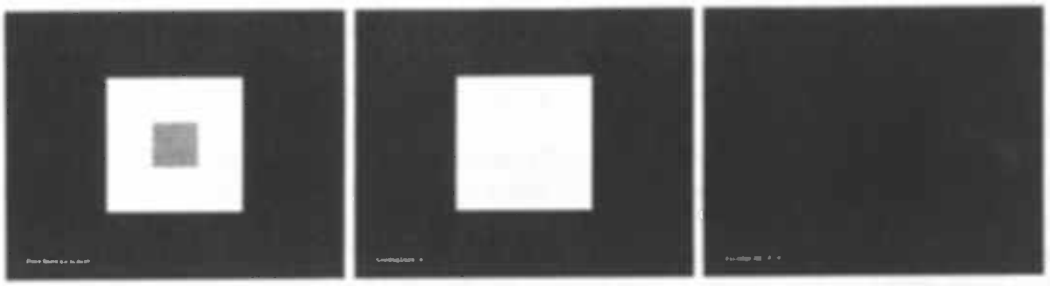
in doubt, consult manufactures instruction manual. Before warming up the computer screen and turning on the computer, hook up the communication cable into the Minolta and the second COM port of the computer. After the warm up period we can place the Minolta in front of the monitor of interest. Arrange the cord with extra length. If there is not sufficient extra length, it may cause connection failure or disconnection. Put the Minolta at a distance that will be desirable when conducting experiments, the minimum distance should be one-thousand and fourteen millimetres as measured from the focal plane indication, with a minimum measurement area of fourteen point four millimetres in diameter. Close up lenses can be attached to the Minolta to measure smaller objects at closer distances. Make sure the Minolta is straight in front of the screen so when looking trough the lens there is no angle between the chroma meter and the monitor.

Before we start the program, first turn on the Minolta in communication mode. This can be done by holding the white F button and turning the Minolta on. This can be checked by looking at the external display. On the right bottom side a 'C' will appear. Starting the program will result in a white square being plotted on the screen with a grey square within the white one. Aim the Minolta in the centre of the grey square to ensure it measures the area where we going to display the different stimuli. When we the see the target area through the viewfinder, focus the lens on the target patch. Focussing the lens on the patch makes certain we measure the target area and gives best measurement results. After finishing these steps we can start the calibration routine by pressing the spacebar.

The measurement program reads in a text file with a range of RGB values and creates from these values stimulus images in centre of the screen. For each stimulus images a signal is send to the Minolta to take a measurement of the current stimuli image. After the program has finished its read in RGB range, all measured data is written out to a file. Before the measurement routine starts, a counter is shown on the screen that gives ten seconds to leave the room before measuring. We will give some more details of program and its measurement step, as well more information on the communication between the program and the Minolta.

The program consists of five stages: `start`, `pre_calibration`, `in_calibration`, `post_calibration` and `exit`. A successful routine will go through all five stages. In the first stage we setup the communication connection between the Minolta and the computer and display a target area on the screen. Communication is done by a RS-232C protocol; `baud rate 4800`, `parity even`, `data length 7 bits` and `two stop bits`, where messages are send and received trough a RTS control handshake, this is request-to-send flow control. The RTS handshake ensures messages are sent and received if both parties are ready for transmission. The program usually issues a command to the Minolta, where the Minolta will receive and process the message. The processed message will resolve in a return a response to the issued command or an error message. The program waits until it gets a reply from the Minolta. If the Minolta has processed all command and transmission issues, it returns to standby modus.

Once the communication is established we send a message to the Minolta to clear its memory and a message is presented on the screen to start the program by pressing spacebar. The program continues after user input. This can be: space bar to continue the program or escape which quits the program no matter what state the program is in. Pressing spacebar launches a countdown timer that gives the user 10 seconds to leave the room and/or switch of lighting in the room. The program is now `in_calibration` mode. Here it will process al the RGB values from the input file onto the screen, where the luminance and chromaticity will be measured. During this stage we



(a) *Startscreen* (b) *Counting down* (c) *In measurement*

Figure B.1: *The figures above show three different stages of the program prior to a measurement. In Figure (a) a grey target is shown at which the Minolta must be targeted. In Figure (a) the program is counting down giving a person to leave the room and/or dim the lights. In figure (c) the program displays a colour for the Minolta to measure.*

will send commands to the Minolta to do a measurement. The `in_calibration` mode consists of three sub stages; `pre_measurement`, `in_measurement`, `post_measurement`. Each of the three sub stages is 2.2 seconds long; this ensures all three stages are given enough time to complete its task. In the `pre_measurement` stage we clear the screen and a RGB value to be measured is presented. Because a new RGB value is issued here, usually some time is needed by the monitor to stabilize the channel output. Two seconds is enough time to get a stable RGB value. In the next stage we send a command to the Minolta to measure the contents on the screen. If the message is successfully received and processed, the Minolta will measure the contents on the screen. Taking a measurement takes 1.6 seconds by the Minolta. When the program has sent its command it will progress to the last stage. Here it will wait for a reply from the Minolta and handle the received data. Messages from the Minolta will give feedback on the quality of a measurement. The program will write the measured data to its memory, but will process the feedback from the Minolta onto the display for the user to read. These feedback messages contain information about the source of error in measurement. These problems can be in the luminance or the chromaticity range, but sometimes if the transmission is interrupted feedback is given about the communication. Because some errors are almost always present when conducting a measurement routine, feedback messages are not written to the output file. For instance measuring low RGB values will give an error message about the low luminance or flickering display of a screen. While this resolves in an unstable measurement these values are taken into account when calibrating a screen. The feedback presented on the screen may give an answer if the results are not within the range of expectations. After all the RGB values from the input file are processed through our measurement pipeline, they are written to a text file. The communication port between the computer and Minolta is closed and a message is displayed on the screen that the program has finished. After the program finished these steps, it will continue to the last stage where the program will shut itself down. The generated output file will contain the presented RGB values and the measured CIE values.

We can use the measured RGB values to create a calibration scheme for the monitor. To be certain that the scheme works we can use the same program as above with some minor alterations to check the calibration. Instead of supplying the program with RGB values, we use a series of CIE values as input.
Now, all the CIE values will be read in and transformed to the appropriate RGB value. This RGB value will then be measured by the Minolta and will be written to a file. The output file will consist of the presented CIE value, the RGB trichromate that creates the CIE value on the screen and the measured CIE value.
From these values we can analyse the calibration scheme and can conclude if the calibration and/or the monitor is appropriate for use in an experiment. The alterations needed for the program to deal with CIE values are: a text file with the calibration scheme and a alter graphics pipeline that takes CIE values as input and transform these to the RGB counterpart. We use the calibration scheme talked about in appendix A and the graphic pipeline from our experiment program. Here we can drop the illumination model in the pipeline and keep the CIE to RGB transform part. Once we measured a monitor and created a calibration file for the new program we repeat the same routine as described above. In Table B.1 an example input and output file is shown.
As a final note we give the reader a piece of MATLAB code that uses the output values from the measurement program and creates from these a calibration file that can be read into the calibration check program or used in our experiment program.

input: *RGB.txt*			output: *result.txt*					
			RGB Values			CIE Values		
			Red	Green	Blue	CIE_x_	CIE_y_	Luminance
0	0	0	0	0	0	0.2770	0.3237	0.385
32	0	0	32	0	0	0.3877	0.3333	0.538
64	0	0	64	0	0	0.4992	0.3361	1.123
96	0	0	96	0	0	0.5621	0.3384	2.295
128	0	0	128	0	0	0.5882	0.3382	4.023
160	0	0	160	0	0	0.6002	0.3381	6.305
192	0	0	192	0	0	0.6071	0.3391	9.611
224	0	0	224	0	0	0.6117	0.3390	14.090
255	0	0	255	0	0	0.6144	0.3393	18.860
0	32	0	0	32	0	0.2808	0.4364	0.888
0	64	0	0	64	0	0.2919	0.5005	2.773
0	96	0	0	96	0	0.2955	0.5230	6.410
0	128	0	0	128	0	0.2966	0.5332	11.440
0	160	0	0	160	0	0.2961	0.5375	18.980
0	192	0	0	192	0	0.2963	0.5397	30.210
0	224	0	0	224	0	0.2964	0.5409	43.210
0	255	0	0	255	0	0.2968	0.5422	58.760
0	0	32	0	0	32	0.2223	0.2530	0.470
0	0	64	0	0	64	0.1798	0.1740	0.788
0	0	96	0	0	96	0.1614	0.1461	1.440
0	0	128	0	0	128	0.1528	0.1353	2.400
0	0	160	0	0	160	0.1497	0.1303	3.670
0	0	192	0	0	192	0.1479	0.1271	5.595
0	0	224	0	0	224	0.1464	0.1256	8.141
0	0	255	0	0	255	0.1462	0.1250	10.640
32	32	32	32	32	32	0.3097	0.3426	1.158
64	64	64	64	64	64	0.3151	0.3541	3.988
96	96	96	96	96	96	0.3158	0.3533	9.528
128	128	128	128	128	128	0.3164	0.3502	17.330
160	160	160	160	160	160	0.3163	0.3528	28.500
192	192	192	192	192	192	0.3153	0.3541	45.070
224	224	224	224	224	224	0.3155	0.3514	65.010
255	255	255	255	255	255	0.3168	0.3545	87.820

Table B.1: *Example in and output file from colour measurement program*

input: *Settings.txt*

red_gun_coeff			
-0.0056	0.0391	0.3653	6.1341
green_gun_coeff			
-0.0014	0.0176	0.3656	5.5010
blue_gun_coeff			
-0.0041	0.0153	0.4335	6.4933
XYZ_to_Lv			
0.7714	-0.4046	-0.0860	
-0.7919	1.4671	-0.0953	
0.0205	-0.0625	0.1813	

input: *CIE.txt*

0.4150	0.3300	30.000
0.3130	0.4320	30.000
0.2590	0.2410	30.000
0.3130	0.3290	30.000
0.4100	0.4600	30.000
0.3100	0.2560	30.000
0.2270	0.3080	30.000

output: *result.txt*

CIE Presented			RGB Values			CIE Measured		
CIE_x_	CIE_y_	Lum	Red	Green	Blue	CIE_x_	CIE_y_	Lum
0.4150	0.3300	30.000	226	125	150	0.4152	0.3274	29.520
0.3130	0.4320	30.000	112	174	116	0.3119	0.4285	28.890
0.2590	0.2410	30.000	166	128	248	0.2556	0.2374	29.430
0.3130	0.3290	30.000	163	151	179	0.3100	0.3233	28.760
0.4100	0.4600	30.000	170	164	50	0.4104	0.4576	28.760
0.3100	0.2560	30.000	203	118	227	0.3056	0.2515	29.810
0.2270	0.3080	30.000	71	165	211	0.2237	0.3002	29.390

Table B.2: *Example In and output file from calibration check program*

```
function [Rcoeff,Gcoeff,Bcoeff,XYZ_to_LV] = create_calib(filename);
%   function [Rcoeff,Gcoeff,Bcoeff,XYZ_to_LV] = create_calib(filename);
%       filename   =   name of calibration file
%         Rcoeff   =   Red coefficients of polynomial
%         Gcoeff   =   Green coefficients of polynomial
%         Bcoeff   =   Blue coefficients of polynomial
%   YXZ_to_LV      =   Transformation matrix from XYZ to Luminance space
%
%   Returns parameters for a CIE 1931 to RGB space routine
[R,G,B,cie_x,cie_y,cie_Y] = textread(filename,'%d%d%d%f%f%f');

dac = R(1:length(R)/3); %set channel cie values.
R_cie = [cie_x(length(dac)) cie_y(length(dac)) (1-(cie_x(length(dac)) +cie_y(length(dac))))]
G_cie = [cie_x(length(dac)*2) cie_y(length(dac)*2) (1-(cie_x(length(dac)*2)+cie_y(length(dac)*2)))]
B_cie = [cie_x(length(dac)*3) cie_y(length(dac)*3) (1-(cie_x(length(dac)*3)+cie_y(length(dac)*3)))]

% get luminance values
R_gun = cie_Y(1:length(dac));
G_gun = cie_Y(length(dac)+1:length(dac)*2);
B_gun = cie_Y(2*length(dac)+1:length(dac)*3);

% dark current correction R_gun = R_gun-R_gun(1);
G_gun = G_gun-G_gun(1);
B_gun = B_gun-B_gun(1);

% use values in the regular interval for fitting,
% this often means dropping low luminance levels
R_index = find(round(R_gun)>0);
G_index = find(round(G_gun)>0);
B_index = find(round(B_gun)>0);

%create 3 degree polynomial function
Rcoeff = polyfit(log2(R_gun(R_index)),log2(dac(R_index)),3);
Gcoeff = polyfit(log2(G_gun(G_index)),log2(dac(G_index)),3);
Bcoeff = polyfit(log2(B_gun(B_index)),log2(dac(B_index)),3);

% calculated luminance values
R_pol=2.^polyval(Rcoeff,log2(R_gun(2:length(R_gun))));
G_pol=2.^polyval(Gcoeff,log2(G_gun(2:length(G_gun))));
B_pol=2.^polyval(Bcoeff,log2(B_gun(2:length(B_gun))));

%create conversion matrix
XYZ_to_LV =   [ [R_cie(1)/R_cie(2)   G_cie(1)/G_cie(2)     B_cie(1)/B_cie(2)];
                [ 1                        1                          1 ];
                [R_cie(3)/R_cie(2)   G_cie(3)/G_cie(2)   B_cie(3)/B_cie(2) ] ];
XYZ_to_LV = inv(XYZ_to_LV);
```

Table B.3: *Matlab example calibration code*

Appendix C

Open Graphics Library

OpenGL or Open Graphics Library is strictly defined as a 'software interface to graphics hardware'. In essence, it is a cross-platform 3D graphics and modelling library that is highly portable and very fast.

OpenGL was first introduced by Silicon Graphics Incorporated as an interface for their workstations. The forerunner of OpenGL was IRIS GL from SGI. Originally a 2D graphics library it evolved into a 3D programming API for that company's high end IRIS graphics Workstations. These computers at the time were more than the general-purpose computers that were on the market, having specialized hardware for fast matrix calculations and hardware support for graphics buffers and other features. SGI being at the top of market at the time realized that it would be good for the industry to help grow the market for high-end computer graphics hardware. A standard would flourish if it would be embraced by a number of vendors which would make it easier for programmers to create applications and content that is available for a wider variety of platforms. SGI realized that software is what sells computers and if it wanted to sell more computers it needed more software that would run on their computers. Other companies realized this too and the OpenGL Architecture Review Board was established in 1992. The founding members of the group were SGI, Digital Equipment Corporation, IBM, Intel and Microsoft, where SGI controlled al the licensing of OpenGL. On july first in 1992 version 1.0 of OpenGL was introduced. OpenGL first became popular in the fields of CAD, simulation and scientific visualization. Later when hardware accelerated cards began affordable for consumer market, the games industry began to embrace OpenGL. At the time OpenGL was a mature and well established 3D rendering API with a strong feature set. Microsoft at the other hand supporting OpenGL to gain some market share from the UNIX platform by making software portable to the Windows NT platform began to develop their own rendering API. Portability was the key factor here; OpenGL being a cross-platform API meant software developers could also port their applications from Windows NT to another platform. Microsoft developed DirectX as an OpenGL counterweight to be used in the game market, claiming that OpenGL was a for precise and exact rendering suited for the scientific and professional CAD and simulation industry and Direct3D for real-time rendering. Here started what people call the API wars. Microsoft and SGI tried to work out their differences and stared a joint effort with Hewlett-Packard with the goal of unifying OpenGL and Direct3D in a new API called Fahrenheit. It initially showed some promise of bringing order to the world of interactive 3D computer graphics APIs, but due to financial constraints at SGI and general lack of industry support, it has since been abandoned. Today OpenGL is accomplished matured API and released version 2.0 being a great step forward in the history of the API. According to the official OpenGL

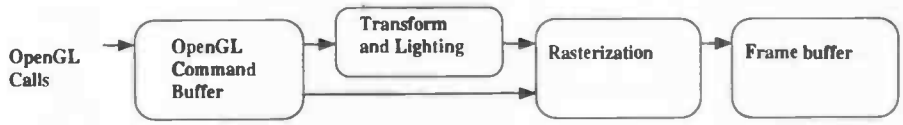


Figure C.1: *Simplified OpenGL rendering pipeline*

website, voting members of the ARB as of November 2004 include 3Dlabs, Apple Computer, ATI Technologies, Dell, Inc., Evans & Sutherland, Hewlett-Packard, IBM, Intel, Matrox, Nvidia, SGI and Sun Microsystems. Microsoft left in March 2003.

OpenGL is a cross-platform rendering library and does not have any I/O capabilities or windows management and should not be mistaken for a programming language. An exception here does exist when we talk about shader programs; these are C like programs that execute on the graphics hardware. Later we will discuss the ins and outs of shader programs. OpenGL can come in two forms: a generic implementation and a hardware implementation. The first is a software implementation that uses a system dependent runtime library that interfaces with OpenGL and the display device. Hardware implementations transform OpenGL commands to an equivalent call to the graphics card. A driver that (usually) comes with a graphics card creates the interface between the hardware and software. Each hardware manufacturer can even supply additional extensions to the graphics API. However to keep OpenGL a standard all the additional functionality to the API is discussed by the Architect Review Board. Overtime when some of these additional extensions become successful they are build into the API. Shader programs are one example of an extension that is added over time. Rendering in OpenGL is done through a pipeline. The OpenGL API can be seen as a state machine, issuing commands to the API leads to some operation or alteration to a stage in the pipeline. The outcome of (subsequent) calls to the API leads to some representation in the frame buffer.
The content of the frame buffer is then passed to the operating systems display application. The OpenGL API works at a low-end therefore operating systems need to have an interface present with the API and a device manager API. Each operating system uses its own graphic user interface protocol and comes with additional libraries to establish a link to the OpenGL API. Linux comes with glX extension for the X11 environment, Apple comes with three extensions for OpenGL; AGL for Carbon applications, NSOpenGL for Cocoa applications and CGL for applications that need direct access to the display. Microsoft uses wiggle or Wgl extensions in their environment. In each environment we can load in the host specific extensions to work with the OpenGL API. In the following chapters we will talk about using extensions and shader programs. We do this by giving an example in a windows environment, because al the applications that were needed for an experiment worked in a Microsoft environment.
First of al we need the correct headers, compiler and runtime libraries. In newer versions of windows they come standard with the operating system. These are generic implementations of OpenGL, usually the graphic hardware manufacturer supplies its own drivers for the API to be hardware accelerated. The needed headers are:

- `gl.h`
- `glu.h`

The compiler libraries in windows are:

- `opengl32.lib`

- `glu32.lib`

and the runtime libraries are:

- `opengl32.dll`
- `glu32.dll`

Often the OpenGL Utility Toolkit is used to handle the device and window management. Being a cross-platform toolkit Glut applications will run almost anywhere, therefore being popular for quickly setting up a OpenGL application. While it is not a standard library for OpenGL, many applications use the toolkit

- `glut.h` : header
- `glut32.lib` : library
- `glut32.dll` : runtime library

These are the standard OpenGL function calls and the OpenGL Utility library and the popular OpenGL Utility Toolkit, this should be old news to OpenGL programmers. For OpenGL to use hardware and platform specific functionality, we need to know which extensions are available for a specific graphic card and sometimes for a platform. Most vendors supply a SDK for their graphic hardware that can be downloaded and show the strength of new features and extensions present in their graphic cards. Popular features that are not present in the current version of OpenGL usually appear in a later version. Because extensions sometimes contain graphics hardware and platform specific features they first have to be passed through the ARB to become standard. Some features will never become standard because they are platform or hardware dependent, but programmers can still use these features to just get that little extra in their application. For hardware manufactures this is a way to test if certain extra functionality becomes popular in the graphic community and to get that little edge over their competitors. In Table C.1 different prefix names from various hardware and software vendors are shown.
To determine which extensions are supported by the OpenGL implementation. We need to pass *GL_EXTENSIONS* to the *glGetString* function. These name strings are generally the name of the extension prececed by another prefix. For core OpenGL extensions this is always *GL_*. If the string is tied to a specific platform (for instance windows), than the prefix will reflect the name of the system in the prefix (*WGL_*). Almost all extensions introduce one or more functions to OpenGL, in some cases an extension may define one or more enumerants. These enumerants are intended to be used in the new functions defined by an extension or sometimes can be passed to standard OpenGL functions. Extensions to functions and enumarants follow the naming convention used by OpenGL, with the exception of a suffix using the same letters as the extension prefix. New extension often do not stand alone and can require the presence of other extensions. Sometimes an extension can even modify or extend the usage of a other extension. Thus it is wise to read the specification and documentation to understand the extension dependencies. New extensions are documented in the OpenGL Extension Registry, which can be found at:: *http://oss.sgi.com/projects/ogl-sample/registry/*
First, to obtain an extension we query OpenGL to return all the supported extension for the current implementation. We do this by making a call to *glGetString* using *GL_Extensions*:

- `char* glExtensionList = (char *) glGetString(GL_EXTENSIONS);`

By making a cast to characters we can use the *c* function *strstr* to search for a extension.

Prefix	Meaning/Vendor
ARB	Extension approved by OpenGL's Architectural Review Board
EXT	Extension agreed upon by more than one OpenGL vendor
3DFX	3dfx Interactive
APPLE	Apple Computer
ATI	ATI Technologies
ATIX	ATI Technologies (experimental)
HP	Hewlett-Packard
INTEL	Intel Corporation
IBM	International Business Machines
KTX	Kinetix
NV	NVIDIA Corporation
MESA	http://www.mesa3d.org
OML	OpenML
SGI	Silicon Graphics
SGIS	Silicon Graphics (specialized)
SGIX	Silicon Graphics (experimental)
SUN	Sun Microsystems
SUNX	Sun Microsystems (experimental)
WIN	Microsoft

Table C.1: OpenGL Extension Prefixes

- `strstr(''GL_<ExtensionName>'',glExtensionList);`

While this sample code works for a greater part of extensions, caution is needed because some extensions may incorperate a substring of the queried extension. This can be avoided by checking the length of the queried extension. When we know a certain extension is available we can request the new extension. In Microsoft Windows this means requesting a function pointer at runtime to the entry of the ICD. First we declare the function pointer:

- `void (APIENTRY * <ExtensionFunction>) (<ExtensionParameters>) = NULL;`

After we have called the adress of pointer we can attempt to assign a entry point to it by using the function *wglGetProcAddress* which is a windows specific function.

- `wlgGetProcAddress(LPCSTR <name of the extension function>);`

When the function succeeds, the return value is the address of the extension function. If no current rendering context exists or the function fails, the return value is NULL. Obtaining an extension is done by performing these two steps.

- `<ExtensionFunction> = (void (APIENTRY *) <ExtensionParameters>) wlgGetProcAddress(''<ExtensionFunction>'');`

To make life for a OpenGL programmer a little easier hardware vendors almost always supply header files with typedefs with most of the extensions out there. The header file *glext.h* contains the generic OpenGL extensions, while *wglext.h* declares all the window specific extensions. Calling an extension function becomes:

- `PFN<ExtensionFunction>PROC <ExtensionFunction>;`
- `<ExtensionFunction> = wglGetProcAdress(''<ExtensionFunction>'');`

If we want to make use of an enumerant we need to specifiy the correct integer to it so OpenGL can recognize the enumerant

- `#define <ExtensionEnumerant> <integer value>`

When we use the headers files this can be omitted, because all the enumerants are then already specified in the header files.
Now that we know how to utilize the extra functionality that is present in OpenGL we can make use of the GLSL (OpenGL Shader Language) functionality that is present in today's OpenGL implementation. Before GLSL programs we had an fixed pipeline, meaning we only could alter some states in the pipeline but each stage would keep its fixed functionality. With the arrival of GLSL programs, parts of the pipeline can be changed to the programmers own needs. There are two types of shader programs: vertex shaders and fragment shaders. Vertex shaders work on vertices and their associated attributes, while fragment shaders work on fragments and their associated attributes. Before we can make use of the GLSL functionality we need to query OpenGL for the existence of this extension. This can be done by querying if the following four extensions are available:

- `GL_ARB_vertex_shader`
- `GL_ARB_fragment_shader`
- `GL_ARB_shader_objects`
- `GL_ARB_shading_language_100`

While this tells us the needed extension are available we now must load in all functions needed for GLSL to work. These functions are:

- `glCreateShaderObjectARB`
- `glCreateProgramObjectARB`
- `glAttachObjectARB`
- `glDetachObjectARB`
- `glShaderSourceARB`
- `glCompileShaderARB`
- `glLinkProgramARB`
- `glValidateProgramARB`
- `glUseProgramObjectARB`
- `glGetObjectParameter(*)ARB`[1]
- `glGetInfoLogARB`

[1] Different parameters are possible

- `glCreateShaderObjectARB`
- `glUniform(*)ARB`[1]
- `glGetUniformLocationARB`

GLSL uses two types of objects: shader objects and program objects. We will first look at the shader objects. Shader objects are attached with shader text and compiled. We can create shader objects through the following function:

- `GLhandleARB myVertexShader = glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);`
- `GLhandleARB myFragmentShader = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);`

When we are done with our shader objects we can delete them:

- `glDelete(myVertexShader;)`
- `glDelete(myFragmentShader;)`

The purpose of shader objects is to attach GLSL code to them and compile the program. To attach GLSL code to a shader we need to specify a pointer to the shader code and attach this to our desired object.

- `GLcharARB *myShaderPtr[1];`
- `myShaderPtr = ShaderText;`
- `glShaderSourceARB(myShaderObject,1,myShaderPtr,NULL);`

The second argument specifies a count that indicates how many string pointers to look for. The last argument tells the length of the read in string, but you do not need to specify a length if a shader text is NULL terminated. When the shader text is attached to an object we can compile the shader.

- `glCompileShaderARB(myShaderObject);`

To query if the attached code is compiled with no errors, we can make use of the functions *glGetObjectParameter*vARB* and *glGetInfoLogARB*. The first commands queries if the supplied code is compiled, the second returns a log with possible errors.

- `GLboolean isCompiled;`
- `GLcharARB infoLog[MAX_SIZE];`
- `glGetObjectParameterivARB(myShaderObject, GL_OBJECT_COMPILE_STATUS_ARB,&isCompiled);`
- `glGetInfoLogARB(myShaderObject,MAX_SIZE,NULL,&infoLog);`

The info log always is always null terminated, so we do not need to supply the exact length of the returned info log.

The second type of objects that are used by GLSL are program objects. These object can be seen as a container for shader objects with the objective of linking them together as a executable. The program object can contain every replaceable part of the OpenGL pipeline. At the moment only vertex and fragment operations can be replaced in the pipeline, but in the future this list could grow. Program objects are created and deleted in the same fashion as shader objects, because there is only one type of program object its entry point does not take any parameters.

- `GLhandleARB myProgramObject = glCreateProgramObjectARB();`
- `glDeleteObjectARB(myProgramObject);`

After we have created our program object, we can assign shader objects to replace parts of the fixed functionality. If we want the old fixed functionality again, we can detach an object from the program object.

- `glAttachObjectARB(myProgramObject,myShaderObject);`
- `glDetachObjectARB(myProgramObject,myShaderObject);`

Before we can use our new GLSL rendering program, we have to link our program object. This process takes each of the compiled shader objects that are in the program container and links them to into a single executable. As with shader objects we can check whether the link was successful and ask for an log.

- `GLboolean isLinked;`
- `GLcharARB infoLog[MAX_SIZE];`
- `glLinkProgramARB(myProgramObject);`
- `glGetObjectParameterivARB(myProgramObject, GL_OBJECT_LINK_STATUS_ARB,&isLinked);`
- `glGetInfoLogARB(myProgramObject,MAX_SIZE,NULL,&infoLog);`

Now that we have linked the shader objects into an single executable we need to validate the program. Validating an program checks some parts of the program that are unknown at link time and are in use when our GLSL program is in runtime.

- `GLboolean isValidated;`
- `GLcharARB infoLog[MAX_SIZE];`
- `glLinkProgramARB(myProgramObject);`
- `glGetObjectParameterivARB(myProgramObject, GL_OBJECT_VALIDATE_STATUS_ARB,&isValidated);`
- `glGetInfoLogARB(myProgramObject,MAX_SIZE,NULL,&infoLog);`

The last step is enabling our GLSL program.

- `glUseProgramObjectARB(myProgramObject);`

If we want to disable our GLSL program we issue the following command.

- `glUseProgramObjectARB(0);`

GLSL programs are very like C/C++, where the main difference are the main data types used in GLSL programs and the standard functions that available. Because rendering calculations often make us of vector and matrix calculations, shader primitive data types include vectors and matrices and subsequent operations on these data types. The other difference are the building data types and functions that can be used or set in shader programs, examples are vertex normals, fragment colour and texture coordinate lookup functions.
Further information about shader primitives and functions can found at:
http://oss.sgi.com/projects/ogl-sample/registry/ARB/GLSLangSpec.Full.1.10.59.pdf
We leave this chapter with an vertex (Table C.2) and fragment (Table C.3) shader used in the experiment program that performs Phong diffuse lighting.

```
uniform mat3        XYZ_TO_LV;
varying vec3        Lv_light;

void main(void)
{
vec3 V          =   vec3(gl_ModelViewMatrix * gl_Vertex);
vec3 N          =   normalize(gl_NormalMatrix * gl_Normal);
vec3 L          =   vec3(0.0,0.0,0.0);
vec3 XYZ        =   vec3(0.0,0.0,0.0);

gl_TexCoord[0]  =   gl_MultiTexCoord0;
gl_Position     =   gl_ModelViewProjectionMatrix * gl_Vertex;
L               =   normalize(gl_LightSource[0].position.xyz-(V));

fraction        =   max(0.0,dot(N,L)))*
                    (step(gl_LightSource[0].spotCosCutoff,
                    dot(normalize(gl_LightSource[0].spotDirection),-L)
                    *gl_LightSource[0].position.w);

XYZ.x           =   (gl_LightSource[0].diffuse.r/gl_LightSource[0].diffuse.g)
                    *(gl_LightSource[0].diffuse.b*fraction);
XYZ.y           =   gl_LightSource[0].diffuse.b*fraction;
XYZ.z           =   ((1.0-(gl_LightSource[0].diffuse.r+gl_LightSource[0].diffuse.g))
                    /gl_LightSource[0].diffuse.g)*(gl_LightSource[0].diffuse.b*fraction);

Lv_light        =   XYZ_TO_LV*XYZ;
}
```

Table C.2: Vertex Shader

```
uniform vec4 RgunPolyCoeff;   uniform vec4 GgunPolyCoeff;
uniform vec4 BgunPolyCoeff;   uniform mat3 XYZ_TO_LV;
uniform vec3 reference;       varying vec3 Lv_light;
uniform sampler2D myTexture;

vec3 CIE2LV(const in mat3 XYZ2LV,const in vec3 CIE)
{
vec3 XYZ          =       vec3(0.0,0.0,0.0);
XYZ.x             =       (CIE.r/CIE.g)*CIE.b;
XYZ.y             =       CIE.b;
XYZ.z             =       ((1.0-(CIE.r+CIE.g))/CIE.g)*CIE.b;
return XYZ2LV*XYZ;
}

float poly(float point,const in vec4 coeff)
{
float polyout     =       0.0;
float log2_point  =       log2(max(point,0.1));
polyout           +=      coeff.w;
polyout           +=      coeff.z*log2_point;
polyout           +=      coeff.y*log2_point*log2_point;
polyout           +=      coeff.x*log2_point*log2_point*log2_point;
return (pow(2.0,polyout))/255.0;
}

vec4 LV2RGB(const in vec3 LVOUT, const in vec4 R, const in vec4 G, const in vec4 B)
{
vec4 polyout      =       vec4(0.0,0.0,0.0,1.0);
polyout.r         =       poly(LVOUT.r,R);
polyout.g         =       poly(LVOUT.g,G);
polyout.b         =       poly(LVOUT.b,B);
return polyout;
}

void main(void)
{
vec3 Lv_out       =       vec3(0.000,0.000,0.000);
vec3 Lv_pixel     =       CIE2LV(XYZ_TO_LV,gl_FrontMaterial.diffuse.rgb);
vec3 Lv_ref       =       CIE2LV(XYZ_TO_LV,reference.rgb);
vec4 tempTexture  =       texture2D(myTexture, vec2(gl_TexCoord[0]));
if (tempTexture.r*tempTexture.g*tempTexture.b==0.0)
{
tempTexture       =       vec4(1.0,1.0,1.0,1.0);
}
Lv_out            +=      ((Lv_light*Lv_pixel)*tempTexture.rgb)/Lv_ref;
gl_FragColor      =       LV2RGB(Lv_out,RgunPolyCoeff,GgunPolyCoeff,BgunPolyCoeff);
}
```

Table C.3: Fragment Shader

Bibliography

[1] Karl-Heinz Bäuml. Simultaneous color constancy: how surface color perception varies with the illuminant. *Vision Research*, 1999.

[2] Paul Bourke. Stereographics. http://astronomy.swin.edu.au/ pbourke/.

[3] Huseyin Boyaci, Katja Doerschner, and Laurence T. Maloney. Perceived surface color in binocularly-viewed scenes with two light sources differing in chromaticity. *Journal of Vision*, 4(2):92–105, 2004.

[4] David H. Brainard. Color constancy. The Visual Neurosciences, Department of Psychology University of Pennsylvania, May 2002.

[5] J. M. Kraft and D. H. Brainard. Mechanisms of colour constancy under nearly natural viewing. *Proc. Natl. Acad. Sci. USA*, 96:307–312, January 1999.

[6] James M. Kraft, Shannon I. Maloney, and David H. Brainard. Surface-illuminant ambiguity and color constancy: Effects of scene complexity and depth cues. *Perception*, 31(2):247–263, 2002.

[7] E. H. Land. The retinex theory of color vision. *Scientific American*, 237:108–128, 1977.

[8] Dr. Lenny Lipton. The stereographics developers' handbook. Technical report, StereoGraphics Corporation, 1997.

[9] Marcel Lucassen. *Quantitative Studies of Colour Constancy.* PhD thesis, Rijksuniversiteit Utrecht, 1993.

[10] Laurence T. Maloney. Illuminant estimation as cue combination. *Journal of Vision*, 2(6):493–504, 2002.

[11] Steven K. Shevell and Jianping Wei. A central mechanism of chromatic contrast. *Vision Research*, 40:3173–3180, 2000.

[12] Mel Slater, Anthony Steed, and Yiorgos Chrysanthou. *Computer graphics and virtual environments.* Addison Wesley, 2002.

[13] H. von Helmholtz. *Handbuch der Physiologischen Optik 2 (Voss,Hamburg)*, 1911.

[14] Charles Wheatstone. Contributions to the physiology of vision part the first. on some remarkable and hitherto unobserved phenomena of binocular vision. *Philosophical Transactions of the Royal Society of London*, 128:371–394, 1838.

[15] Charles Wheatstone. Contributions to the physiology of vision. part the second. on some remarkable and hitherto unobserved phenomena of binocular vision (continued). *Philosophical Transactions of the Royal Society of London*, 142:1–17, 1852.

[16] Yasuki Yamauchi and Keiji Uchikawa. Depth information affects judgment of the surface-color mode appearance. *Journal of Vision*, 5(6):515–524, 2005.

[17] Joong Nam Yang and Laurence T. Maloney. Illuminant cues in surface color perception: tests of three candidate cues. *Vision Research*, 41:2581–2600, 2001.

[18] Joong Nam Yang and Steven K. Shevell. Stereo disparity improves color constancy. *Vision Research*, 42:1979–1989, 2002.