University of New South Wales

Polytechnic University of Valencia

Rendering stereographic 3D images in cylindrical spaces.

Realized for the IARFID Master by:

Rubén Díaz Hernández

(rudiaher@gmail.com)

Directed by:

Ardrian Hardjono (UNSW)

Co-directed by:

Francisco J. Abad Cerdá (UPV)

iCinema Centre for interactive cinema research

September 2010

Declaration

I declare that the work in this Masters thesis has not previously been submitted for a degree nor it has been submitted as part of the requirements for a degree except as fully acknowledged within the text.

I also declare that this report has been completely written by me, and the only aid that I have been given has been for issues of translation. Any other help that I have received in my research and development of the system proposed has been acknowledged.

In addition, I declare that all the information sources and literature used are referenced in the bibliography.

Rubén Díaz Hernández

Acknowledgements

First, I would like to thank my supervisor Ardrian Hardjono for all the support and advice through the development of this Masters thesis. I would also like to recognise the rest of the *iCinema* Research Centre staff, for supporting me with the problems that I have encountered along the way. I will be leaving *iCinema* after a short experience of 6 months, and I am aware of how much I have learnt during my stay and how important this experience will be for my professional future.

I would also like to recognise my co-director in Spain, Paco Abad. His suggestions and support have been very much appreciated.

I would also like to give special thanks to Vanessa, for her help with translation, but especially for being there these past 6 months.

And last, but not least, I would also like to thank my family, that has been supporting me from the other side of the world. Para tí, abuelo.

Abstract

Immersion: *State of consciousness where an immersant's awareness of physical self is diminished or lost by being surrounded in an engrossing total environment, often artificial.*

One of the most significant characteristics of the human being has always been the power to reproduce environments, concepts or situations that do not correspond with the present moment or location. The interest shown by people for doing this has been of a very different nature; motivations include religion, art or entertainment.

Probably the earliest examples are the very first carvings and paintings engraved in caves by prehistoric people. Writings, paintings and sculptures are other examples that have been in pursuit of the achievement of similar goals throughout human existence. Another interesting mode of the '*substitution of reality*' that we can consider, and that has always been carried out by different civilizations to the present day, is theatre. The word itself derives from the ancient Greek*, theatron* (*θέατρον*) meaning 'the seeing place'. The idea is that the audience is abstracted from what is occurring outside of the theatre to become *immersed*, for a brief period of time, in an alternative succession of happenings performed by the actors playing their roles.

Translating this concept to the modern era and the use of new technologies, we can find multiple types of systems involving different levels of immersion for the user. In our homes we can now find anything from movies to computer games with high end computer graphics, where interaction also plays a decisive role. Presently, we can even acquire our own 3D televisions and sophisticated surround sound systems. In other contexts we find more complex cinema screens

(conventional and 3D), but also virtual reality systems and *HMDs* (head mounted displays) that can change the total or partial environment that is being perceived by the user.

Nevertheless, when it comes to displaying 3D worlds on flat surfaces there are huge sacrifices that have to be made. With the classic projections that are widely used (and the standard model for most systems) we find many problems in trying to take the experience to a high level of immersion. The reason for this is that the nature of these projections does not correspond with the way we perceive the world. In an extremist approach, the full immersion and total abstraction from the real world requires a believable scenario that is able to substitute what our senses would usually perceive with the new stimulus. Of course, in order to achieve this goal (if we are to omit taste and smell) we need to introduce the use of haptic devices that provide feedback to our sense of touch, but the approach that has been more widely studied only involves the substitution of sight and hearing. In that framework, we still need to understand how signals are physiologically received by our body and attempt to reproduce them in the most accurate way. The output of the system is represented on the '*barrier*' that separates the user from the real word. Often this barrier is a screen that presents the synthetic world to the user. Usually, cylindrical or spherical screens (i.e. domes) are used, because unlike the flat surfaces systems, they have a seamless and continuous surface.

It is the aim of this thesis to discuss and present a method to immerse the user in an artificial world within the context of a cylindrical screen of 360 degrees (*AVIE*, *Advanced Visualization and Interaction Environment*). For doing so, a full implementation of the system is given and the specification of a new rendering method for cylindrical (or spherical) spaces that provides real omnistereo (stereographic images correct for every direction) is proposed and integrated into the system itself. The system is to be used in practice in an interactive display for the Victoria Museum of Melbourne (Australia).

Table of contents

Chapter 1

Introduction

1.1 Problem overview:

Day after day, people expect more and more when interacting with a computer and take to the extreme the feeling reported by multimedia contents whether they are games or movies. Technological advances make the experience something more and more enjoyable, intuitive and realistic. Naturally, in synthetic images the better the systems (and in particular the graphics cards) the more "real" they will look because more calculations can be processed per each frame.

However, when it comes to the enhancement of the immersion of the user in the artificial world, it is not only the quality of the graphics or the definition of the videos that is important but there are other factors that become crucial as well. These factors are related with the way the information is presented to the user and how interaction with the system is conceived. Ideally, the user will interact naturally and in a very intuitive way with the computer and feedback from the system will be convincing and in harmony with the artificial environment. There is a field of study that deals with interaction between people and computers, known as *Human-Computer Interaction* (*HCI*). As it studies a human and a machine in conjunction, it requires supporting knowledge on both sides: On the machine side techniques in computer graphics, operative systems, programming languages and development environments are relevant. On the human side, communication theory, graphic and industrial design disciplines, linguistics, social sciences, cognitive psychology and human factors are to be considered. The *ACM* (*Association for Computing Machinery*) defined *HCI* as "*a discipline concerned with design, evaluation and implementation of interactive computing systems for human use and with the study of major* *phenomena surrounding them*". Special attention to *HCI* is important because poorly designed human-machine interfaces can lead to many unexpected problems, frustration and lack of satisfaction of the final user, destroying the natural flow of information between both parts and the efficiency of the task.

As a consequence of the above, if the goal of the system is to recreate a virtual scenario and make the observer feel a part of it, it no longer makes sense to present a 3D scene to the user in a flat 2D screen even if we are using stereographic images. The reason for this, is that the nature of our eyes works in a very different manner to the one expressed by the mathematics contained in flat perspective projections. In another words, it is technically impossible to represent the view of a scene as we do on a flat surface using projective matrices, and consequently, using that approach the realism is lost. On the top of that, when it comes to using surrounding screens, the use of very wide view angles brings a lot of new problems to the task since we are presenting to the user a view volume much greater than what can be seen at once, and that is something that has to be done with care.

1.2 Objectives:

The goal of this master's thesis is the implementation of a render system for curved surfaces offering accurate 3D for an unlimited width angle.

These features are needed to represent panoramic scenes onto seamless curved screens. Usually the surface to be projected can be considerably large so high definition images need to be calculated. In order to do that in real time, it may be necessary to split the task between several machines (a cluster of computers). In addition to this, other desirable properties can be obtained when using multiple projectors as explained later in this report.

The area of application of this kind of system is very broad. It can be summarized as comprising all applications that are needed to immerse the user into an alternative scenario. The level of immersion (abstraction from the world) can vary from one application to another. It can be particularly useful in contexts like scientific visualization, *Computer-Aided Design (CAD)* or the entertainment industry.

To achieve this, it has been necessary to investigate how the best and most accurate result can be obtained without sacrificing computational efficiency. In order to do that, it has been essential to analyze multiple techniques used in similar frameworks and discuss their strong and weak points. During the process, many different approaches and configurations have been considered as they will be presented throughout this report.

The most important features of the system are described below:

- Display of wide angle view volumes calculated for the point of view of the observer. This implies that if a change in the position is registered the new view volumes will change in consequence according to the user's movements.

- The scene is calculated independently for each eye and the stereo is valid for the whole extension of the screen.

- A fully featured calibration system is proposed. The goal of this program is to calibrate projectors so they can properly display the scene on the curved surface that is to be used. For doing so, two steps are clearly differentiated; warping of the display mesh and blending. Both stages have been implemented to be simple to use and offer great flexibility and so the calibration process becomes quick and easy. The blending is necessary if several projectors are used and a seamless projection is desired.

- A clustering mode is implemented so several machines can run the same application simultaneously and frame synchronized. The implementation of a server that controls the slaves is also provided.

- Multiple utilities also implemented and fully included in the cylindrical system, such as incorporation of images, video and animations (synchronized with the other slaves), meshes, text, materials, panoramic images, billboard objects, *BSP*'s (*Binary Space Partition*), Paged Geometry and so forth.

- Illumination techniques (i.e. *per pixel Phong shading*), *bump mapping*, and texture mapping perspective corrected for cylindrical spaces.

- Implementation of the system done using a widely used 3D engine (*Ogre3D*) offering the possibility of an easy adaptation of any application for *Ogre3D* to work in a wide angle curved screen. Due to its modular nature (object oriented) the implementation of the system is easily accessible by anyone.

- The user defines the scene in a traditional *Ogre3D*'s way without having the concern of additional transformations because is defined in 'real world' units (meters) and displayed correctly on the screen.

- Two different approaches of the wide angle scene provided; *Slice-based* and *Shader-based*.

- The *Shader-based* approach avoids discontinuities (because the concept of slices does not exist) and a long list of problems derived from that approach is avoided at the same time that efficiency is improved.

The system has been developed and tested for the *AVIE* system (even though it can be adapted very easily to other systems such as the *iDome*) at the *iCinema* centre. For that reason its capabilities can be easily improved just using other features of the *AVIE*. For instance, the tracking system or the input devices (as the joystick or the orientation sensor).

1.3 Preliminary description:

Even though the renderer presented can work for other systems, the description for the *AVIE* (*Advanced Visualization and Interaction Environment*) is described below:

The first necessary step is the calibration for the projectors including the blending. As a result from that process we obtain the calibration files that will contain all the necessary information. Once that is done and the scene is defined in our *Ogre3D* application, the computer running the server program has to distribute all the necessary files for every computer (associated with each projector) using the option for that purpose. Once the distribution has finished, the server can initialize all the clients. Using the information obtained from the calibration, the projectors will show the corresponding portion of the scene in the proper area of the screen with the blending masks applied, (that is for both eyes, the stereographic illusion can be observed if the users wear polarized glasses). With the application running, every client (computer) will be synchronized with the rest just by asking the server when they can swap the display buffers. This synchronization will also be applied to objects in the scene, for example, animated meshes and videos will show the frame corresponding to the present moment.

1.4 Description of the report structure

For the organization of the information presented in this report, the structure that is explained below has been chosen. Firstly, there is an introductory chapter (the current one) in which the problem is presented as well as the project itself. In it, the main objectives to be obtained have been commented on, so that the range of possible applications becomes clear. In the same way, a brief explanation of the implied parts has been given.

The purpose of the following chapter, Background, is to show the state of the art of similar immersive systems as well as the techniques and existing hardware and software, highlighting the main advantages and disadvantages. Some of these have provided inspiration and reference for this proposed system.

The Analysis chapter discuses the process of investigation and testing that led to the proposed solution. In this chapter there is a definition of the overall behaviour of the application and the requirements that have to be fulfilled for this thesis.

The intention of the following chapter is to describe the design stage (and consequently how things have been done). The main modules, classes and so on, will also be described. The development methodology employed and an in-depth description of the solution will be described as well. This is essential in order to understand the flow of information that moves within the system and the algorithms employed.

In the Results chapter, there are some complete examples of use and the obtained results. Considering these examples, the system will be evaluated to demonstrate that the goals proposed have been satisfied.

Finally, the last chapter will describe the course of development of the process, in terms of deadlines, the main problems that had to be faced, and so forth. Also, in this part the main conclusions that can be extracted from the work are presented, as well as possible future trends, extensions, improvements or variants of the final system.

Chapter 2

Background

The main key concept of this master's thesis is "*immersion*". In a wide variety of situations it is desirable to make the user feel part of an artificial environment. Nevertheless, the level of the immersion can vary notably depending on the intention of the application or the technology in use. Sometimes the system attempts to isolate the observer from the real world, substituting the contents perceived in reality by alternative ones, while other times elements from the real world are combined with some others that are not physically there (virtual). Taking this into account, it makes sense to define a *reality-virtuality continuum* as previously defined by some authors (Milgram 1994 [1]), as shown in the image below (Fig. 1)

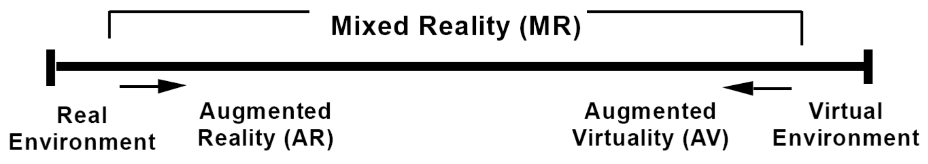


Fig. 1: Reality-Virtuality (RV) Continuum. (Milgram 1994 [1]).

As it can be deduced from the figure, the perfect abstraction from the world is represented in the right part of the continuum. Ideally, the user will perceive the virtual world in first person and the interaction with the environment will be absolutely natural and intuitive. The real world will not be perceived by the user, only the virtual one. This implies the substitution of all the major senses (sight, hearing, taste, smell and touch). Feedback to the actions of the user will also be perfectly convincing. Obviously, a system with those characteristics is technically impossible to implement and rather utopic. Leaving philosophic questions about the meaning of "*reality*" aside, this is similar to what has been shown in science fiction movies such as "*The Matrix*" (1999) and many more in which people connected to the system are not even aware of the existence of the real world.

More plausible systems can be located in the middle regions of the *continuum*, taking us from our real world (on the left extreme of the fig.1) all the way to the virtuality region. Considering only the visualization part of the immersive system, (because that is the purpose of this thesis), some of these systems will be based in *HMD*s (head mounted displays) to ensure the user perceives the world in first person, transparent *LCD* screens (to combine real and virtual images), or projected screens surrounding the user.

Multiple possibilities are available and some provide a higher level of immersion than others, even though there are other important factors to consider (such as the interactivity that is permitted or the number of users that the system supports simultaneously). For that reason, in this chapter a summary of the state of the art and a survey of the most important technology that can be used in immersive visualization systems are given.

2.1 Technology

2.1.1 Stereographic 3D images

In order for either synthetic images or recorded video to present information in a realistic way to the user, it is necessary to create the illusion of three dimensions. It is often not enough to calculate the scene for a unique point of view since the way humans perceive depth is based on the fact that we see the world from two different perspectives (one for each of our eyes). There are other important factors and a many details to mention regarding this (a more in-depth analysis of the nature of the human sight will be included in the Analysis chapter), but for the moment the present concern is the stereo pair itself and how to present the information to the user. There are multiple options, but not all are suitable for every purpose. Flexibility and questions of economy matter as well.

2.1.1.1 Stereoscopic binocular systems

A binocular system is a device that isolates the information that the eyes of the observer receive from the real world, substituting it with an alternative in the virtual world. This kind of device completely covers both eyes with two screens that show the scene under different perspectives. Doing so, if done properly (which means that the perspectives are calculated for each eye), our brain can process the information, accepting the trick. The users perceive a 3D image, but what they are actually seeing are just two 2D projections, only slightly different.

There are many different types of binocular systems in the market; the most commonly used being the *HMD*s. As the name indicates, these devices are held to the head of the user, with all the advantages and disadvantages that this entails. Another type of binocular system is the *BOOM* (Binocular Omni-Orientation Monitor) by *Fakespace* labs. Unlike *HMD*s, they are not directly worn by the user but held by a mechanical jointed arm that the user can move. They usually have a counterweight so the user does not feel so connected to the system. The user observes the virtual world though a carcass that is attached to the end of the mechanical arm, where the two small displays are located. An example of each is shown in fig. 2.

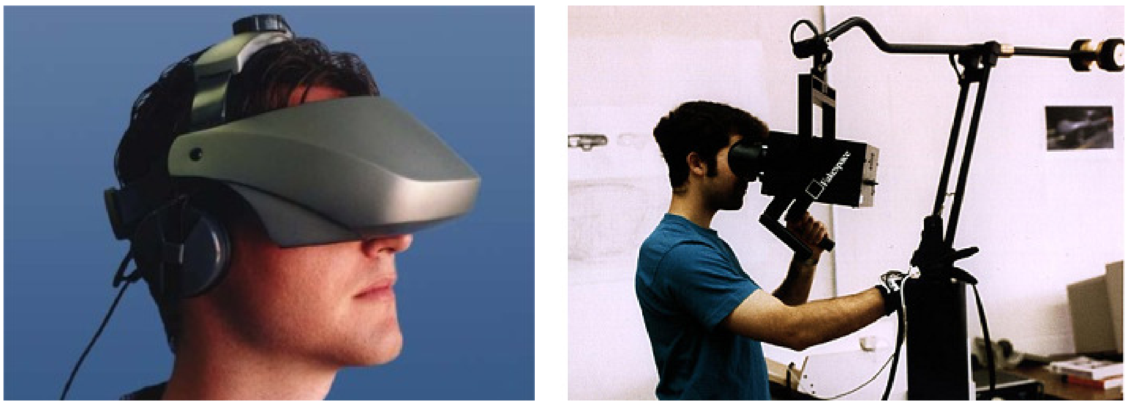


Fig. 2: *HMD* by *5DT* (on the left) and *BOOM* by *Fakespace* (on the right).

Despite being monocular rather than binocular, the first devices appeared in the mid seventies designed with military purposes. They were at times extremely heavy and it was not until the *CRT* (Cathodic Rays Screens) technology gave way to the *LCD* (Liquid crystal display) that these systems became relatively comfortable to wear. Similar devices have been used also for augmented reality applications, (also with transparent *LCD* screens), with one and two displays. The fact that the device is attached to the user´s head, simplifies head-tracking, which can be advantageous if a specific point of view in required. However, many drawbacks also exist. Technology limits these devices and, in general, they restrict the movements and isolate the user too much. The quality of the display screens is also an issue; normally flat screens with very limited resolution are used. Depending of the purpose of the application these disadvantages can be sufficient in determining the decision to use another alternative. The aforementioned, together with the prohibitive price, translates into the fact that these devices fall into disuse for most of the applications for the general public, as for example, in entertainment.

2.1.1.2 Autostereoscopic displays

Autostereoscopic systems include all devices that display stereoscopic images without the use of any special lighting conditions, headgear or glasses. In order to achieve this, it is necessary that the display itself shows each eye a different image. The two images (for left and right eyes) are codified by being interlaced in columns and a type of barrier is located in front of them, so that each eye only sees the corresponding columns that form the final image. *LCD* screens are used in this case because, unlike *CRT* screens, *LCD* pixels have high positional stability. This basic idea is shown in fig. 3 where the red regions of the image correspond to the left eye and the blue ones to the right.

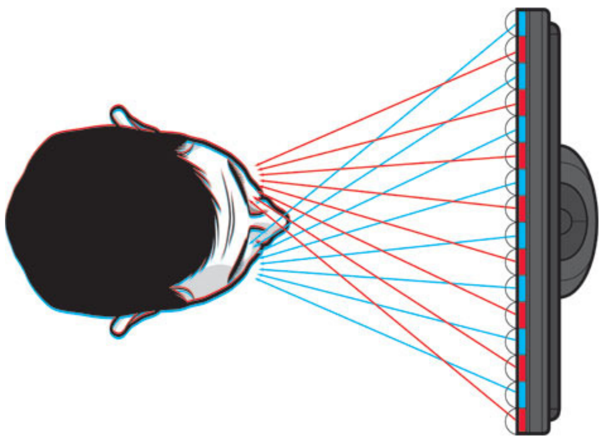


Fig. 3: Autostereoscopic screen with two images encoded and correspondency with eyes.

There are two main designs, parallax barrier and lenticular. The first, locates a parallax barrier (which consists of multiple parallel vertical opaque bars) on top of the interlaced image in a manner that the parts of the image that are to be obscured from the view of the opposing eye are covered (see fig. 4a). In more complex designs, there are more than 2 images 'hidden' behind a barrier so that the user may move left and right and continue to see how the image changes. The lenticular method uses cylindrical lenses aligned with the columns of the *LCD* pixels instead of optical apertures. The idea is that these lenses direct the diffuse light from each pixel so it can only be seen within a limited angle in front of the display. Consequently, this allows different pixels to be directed to either the left or right viewing windows (see fig. 4b). The viewing windows define those areas for which the optical illusion is correct. In both the parallax barrier and lenticular methods these windows are not very wide, being one of the most significant drawbacks of this technology.

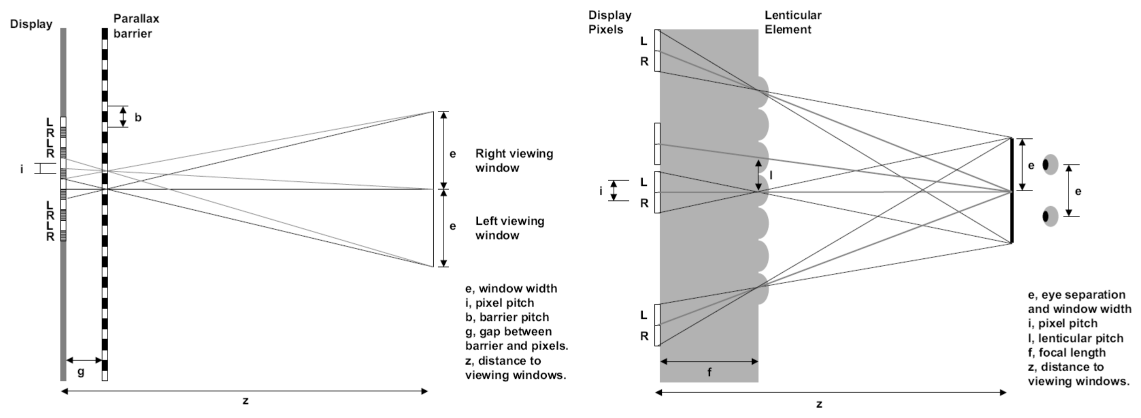


Fig. 4: Detail of a parallax barrier method on the left (a), and lenticular method on the right (b). (Holliman 2005 [2]).

Screens can be purchased with a stereoscopic filter or separately if they are compatible with each other. As mentioned before, they are prepared for showing the image accurately within a very restricted area. If the user moves out of this region, the image becomes fuzzy or deformed, losing the 3D appearance. Another negative point is that as two or more images have to be represented in the same area, the resolution is substantially reduced (typically halved). Nevertheless, this is a very interesting stereoscopic system for some applications, especially the ones designed for single users that are required to be limited in movement. One example is the portable console *Nintendo® 3DS* that features an autostereoscopic parallax barrier screen with a resolution of 800x240 pixels (400x240 effective for each eye).

Both systems only display horizontal parallax. There is a similar display type that is able to produce both vertical and horizontal parallaxes, (which suggests that the user can move left and right but also up and down and see the different sets of images). This system is called integral photograph. This kind of display uses spherical lenses (instead of cylindrical), to present horizontally and vertically varying directional information, producing a full parallax image. In fig. 5 an integral image’s spherical lens array is shown, illustrating how the light rays are diffracted.

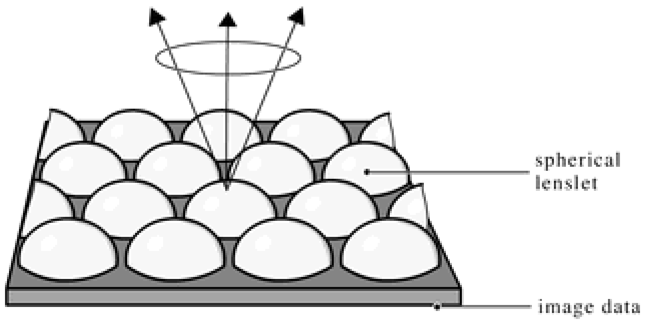


Fig. 5: Scheme of an integral photograph display. (Halle, 1997 [3]).

This type of lenticular display is even less common than the cylindrical lenses; mostly because their spatial resolution is sacrificed to directional information.

One technology that may be of interest for future development is based on holographics, first discovered by Dennis Gabor. The way in which holographic stereograms overcome resolution constraints (and also problems of repetition due to crosstalk between adjacent display elements) is based on the way information is formed within the display. In this case the process uses optic techniques (electron waves rather than light waves), and records the information microscopically as fringe patterns. The details of the holographic stereogram's recording process are beyond the scope of this text and only a brief comment and bibliographic references are given (Benton, 1982 [4]; Graham (2003) [5]). Within the last decade or so, typical holographic stereograms have become very popular and easy to produce, making them suitable for stamping and publication. However, they are not appropriate for dynamic imaging due to long exposure times that they require to be imprinted and the fact that most of the materials employed are non rewritable. Elecro-holography is the application of holographic techniques to electron waves and may be used together with optical computation systems to create dynamic 3D holographic images. This could be one of the upcoming technologies for displaying dynamic auto-stereoscopic images in the future. However, the large volumes of information needed to be loaded on the displays in real time (multiple points of view are to be supported), and the current technology prevent this approach from producing high quality three-dimensional images using affordable hardware anytime within the near future.

2.1.1.3 Volumetric displays

So far, the systems discussed in this report are systems that show the image on a flat surface and simulate depth through the disparity between the channels for left and right eyes. On the contrary, a volumetric display device is a graphical display that forms a visual representation of the objects to be displayed in three physical dimensions. For that reason, these are not purely stereoscopic as the previous ones. They do not show the scene as a pair that creates the illusion, and do not require the use of any additional device to see the 3D (it is real 3D) and therefore they have been included in this chapter.

The philosophy behind these kinds of devices is radically different and the techniques employed, as well as the difficulties encountered, have a completely different nature than those based on stereoscopic pairs of images (Blundell & Schwarz, 2000 [6]). Using this new technology, a first attempt has been made to display very basic images. Better quality images are not yet able to be produced due to technological restrictions. The most significant difference is the fact that objects displayed are meant to be within physical space, which means they are shown located in a volume predefined by the display. This implies that 3D projections onto 2D surfaces as we are used to see in movies are not possible in volumetric displays. If it were possible to translate the perspective view volume to the 3D space of the display, severe distortions would occur. This proves this type of device very suitable for many applications such as scientific visualization or medicine, while perhaps, not appropriate for some others. Currently there are two main groups of volumetric displays: static volume and swept-volume display.

The first is probably the most 'direct' form of volumetric display, since it defines an addressable space constituted by elements called *voxels* (as opposed to pixels) that are transparent in 'off' state, and become luminous when they are 'on' (fig. 6a). In order to do this, laser light technology is usually employed to encourage visible radiation in a solid, liquid or gas (ideally air). For example, one of the techniques (Payatagool, 2007 [7]) uses a focused pulsed infrared laser to create balls of glowing plasma at the focal point in normal air. Each pulse creates a popping sound, so the device crackles as it runs. It is thought that this system can be scaled to any size, allowing 3D images to be generated in the sky. Nevertheless, the resolution and colour that can be displayed, as well as the definition are still relatively restricted. These kinds of devices represent a new set of possibilities yet to be explored.

The second display that generates “real” 3D images is the swept-volume (Jones, 2007 [8]) (fig. 6b). These devices rely on the human persistence of vision to fuse a series of regions of a 3D image displayed individually. The most common design is the revolution projected screen. The image is decomposed in “slices” that are projected onto a display surface undergoing motion. The projection changes as the surface rotates. Due to the persistence of vision, humans perceive a volume of light. The surface can be reflective, translucent or a combination of both. The main problem of these displays is that the image must be refreshed at a minimum frame rate for every slice. Usually, to create a consistent 3D image, a large amount of slices are necessary. This implies that the image needs to change very quickly, so very high frame-rates that cannot be handled by some projectors are required. It is also necessary to implement the system following hard real time restrictions to properly synchronize the output of the computer with the physical position of the surface.

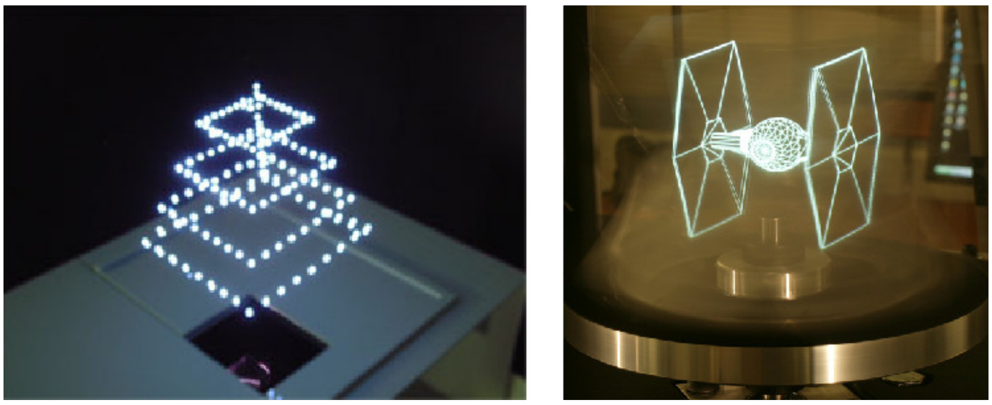


Fig. 6: Volumetric displays: Static volume on the left (a) (Payatagool, 2007 [8]); Swept-volume on the right (b) (Jones, 2007 [7]).

2.1.1.4 Stereoscopy with glasses

Another approximation that is used to achieve a 3D illusion with two flat images consists in locating a filter that only allows the corresponding image to be viewed by each eye, in between the screen and the observer. Nowadays, this is the most common approximation that is used even though several methods can be differentiated and organized in two general groups: active (if the glasses have electronics that interact with a display) and passive systems (if they have not).

2.1.1.4.1 Passive systems

The name ‘passive’ comes from the fact that the filtering process is done needless of any communication between the glasses and the main system. For this reason, the filter for each of the eyes and the images prepared for each needs to be different. One of the most traditional is filtering based on colour. This approach was very commonly used a few years ago because of its simplicity (needless of special displays or projectors) and inexpensiveness. The 3D effect was achieved, solely by using colour filters (i.e. red and cyan) for the eyes and displaying the images masked by the corresponding colour (fig. 7a and 7b). The image product of the superposition of the two images for each channel is known by the name of anaglyph and was used for the first time in 1853. However, the quality if the image is destroyed and loses its natural appearance due to the colour filtering, as well as the very obvious and undesired *ghosting* effect obtained (the colour filters are not able to neutralize completely the image for the opposite eye and some silhouettes are clearly visible). There have been several refinements to the basic concept and another technologies based on colour filtering have improved in some way the basic anaglyph results (*ColorCode3D*, *ChromaDepth* and *Anachrome method*). Colour filtering is still used nowadays because it does not require any specialized device apart from the glasses, which is its greatest advantage.

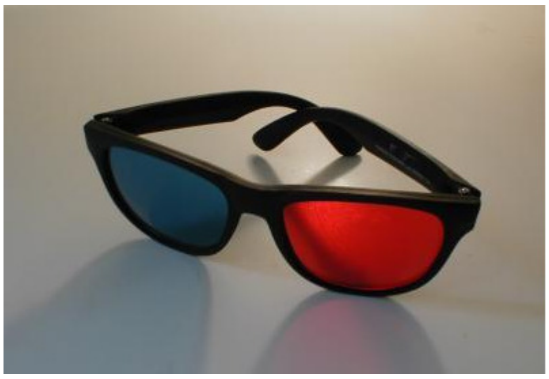
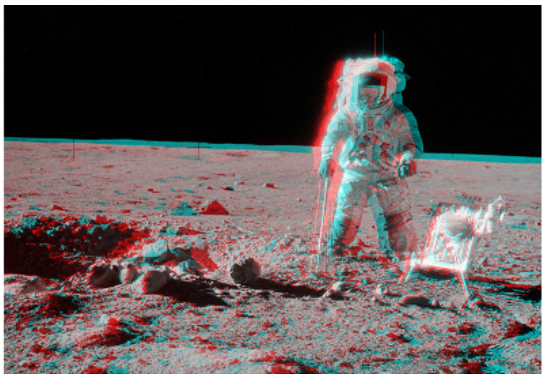



Fig 7: Anaglyph glasses on the left (a) and anaglyph image on the right (b).

Another possibility consists in polarizing light waves. Light in its natural state is not polarized, but passing it through light filters so the shape of the waves become~~s~~ coherent and can be filtered easily by glasses, which has become a very usual technique due to its simplicity (fig. 8). Typically this requires two light emitting sources (normally projectors) with filters provided. The most common way~~s~~ of polarizing the light is linearly and circularly. In case where polarization is produced linearly, light is polarized vertically for one eye and horizontally for the other. The filter will let similarly polarized light pass and block orthogonally polarized light. The effect begins to cease as the user tilts the head, therefore users are required to keep their heads vertically. This restriction can be eliminated by using circularly polarized filters with opposite handedness. Light that is left-circularly polarized is screened by the right-handed filter and vice versa. Although the images still remain associated to the eyes without head-tracking techniques, the parallax exhibited by the stereographic image would no longer be coherent to the eye's position (horizontal distance inter-ocular of 6 centimetres is assumed normally when there is no tracking).

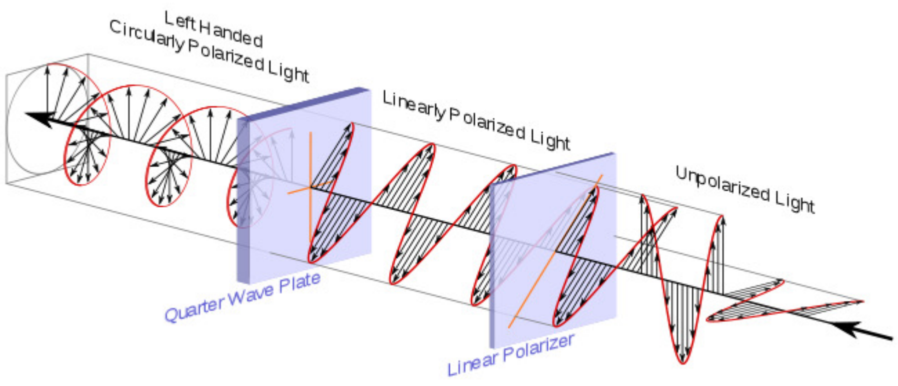


Fig 8: Polarization of light.

Compared to anaglyphs, polarized images produce a full-colour image which is considerably more comfortable to watch, as the *ghosting* effect is also significantly reduced. The price of these glasses is slightly higher, but the major problem is that they require a special set up that uses two synchronized projectors and projection on an appropriate surface (typically a silver screen that preserves the polarization) as most typical televisions cannot support polarization.

Another interesting technology worth comment (regarding passive systems) is *Infitec*^{TM} (Helmut & Fritz, 2005 [9][10]). The name stands for *Interference Filter Technology*. Basically, it consists in multiplexing wavelengths. The range of visible light is separated into six narrow bands, two in the red region, two in the green and two in the blue (R1, R2, G1, G2, B1, B2). The triplet of wavelengths (R1, G1, B1) is used for one eye, while (R2, G2, B2) are used for the other (see fig. 9). As the human eye is largely insensitive to such fine spectral differences, this technique is able to generate full-colour 3D images. This technique has been also called '*super-anaglyph*' because it is a sophisticated form of spectral-multiplexing, which is exactly the idea followed by the anaglyphs. *Dolby*^{TM} adopted this technology for its *Dolby* 3D theatres. In the figure below (fig. 9) it can be seen how an *Infitec* system works.

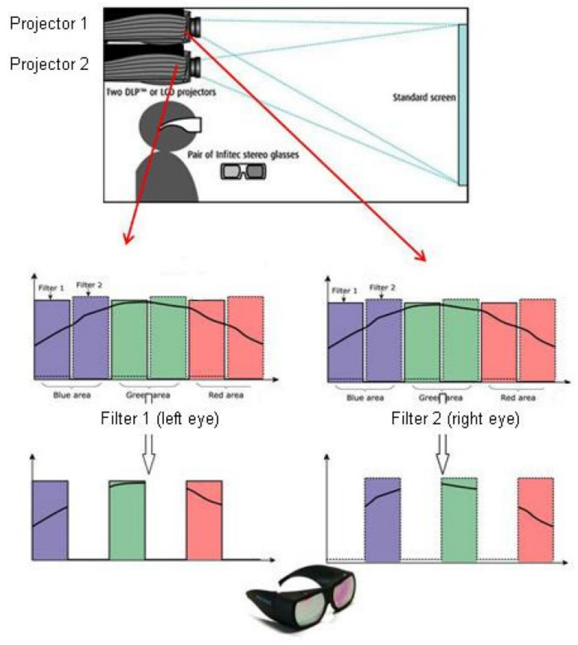


Fig 9: Detail of how *Infitec* works.

To conclude with this summary of passive glasses systems, it is worth mentioning that the exploration of new methods have sometimes been more directed by physiological aspects rather than technological. A clear example of this is the *Pulfrich* effect, a psycho-optical phenomenon that can be used to create a 3D visual effect. The effect is generally induced by placing a dark filter over one eye, but can also occur spontaneously in several people with eye diseases (i.e. cataract). The widely accepted explanation of the apparent depth is that a reduction in retinal illumination (compared to the fellow eye) implies a delay in signal transmission, inducing an apparent spatial disparity in moving objects translated by our brain as depth information. It is assumed that visual system latencies are shorter for bright targets compared to dim ones. The effect has already been exploited in some films, television advertisings and game

3D presentations. This illusion can obviously be combined with many more of the methods already explained.

2.1.1.4.2 Active Systems

These systems are called ‘active’ because they need to receive information from the display to function, and thus they need an additional power source to be able to process the input. The way in which the different images for each eye are displayed differs from the passive systems in the fact that they are not shown simultaneously. Consequently, the glasses have to block or let light through in synchronization with the images on the computer display, using the concept of time-division multiplexing (see fig. 10). The synchronization can be achieved using an infra-red signal, *Bluetooth*, radio frequency, *DLP-Link* and so forth.

In order to obtain the desired effect, a transparent glass that can become opaque is needed. LCD displays have that property, and thus they are used as the glasses (*LCD shutter glasses*). Whether the glass lenses appear opaque or not depends on the voltage applied to them. The illusion works due to the persistence of vision that creates the feeling that there are not blocked frames, and it is perceived as a whole sequence. This phenomenon has already been mentioned before in this report (volumetric displays). And according to that theory, the perception process of the human brain and retina work keeping the image for a brief period of time (just like a buffer) which explains why the animations are perceived as a whole succession when a set of images are shown instead of isolated images.

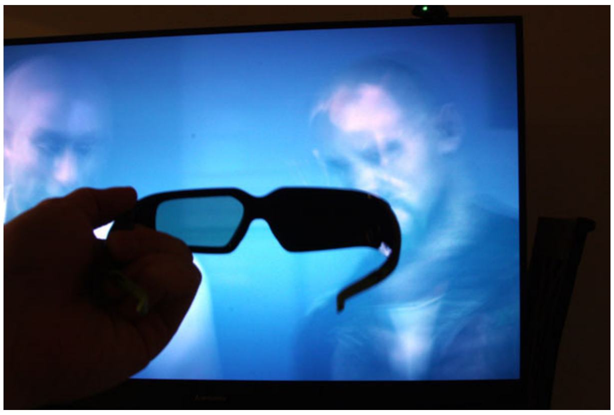


Fig 10: ***Shutter glasses*** **working. Only one of the LCD glasses lets light through for a given instant.**

The frequency of which the image swapping is perceptible depends on the brightness but it is generally around 16 images per second, which is considered the lowest frequency that humans are able to perceive continuous movement. In this case much higher frame-rates are needed (generally not less than 120 hertz, 60 for each channel). The feeling the user has while they are watching the screen is similar to watching a projection wearing sunglasses while without the glasses the images appear to be blended.

On the one hand, the ghosting effect is mostly eliminated and as the glasses are colour neutral they enable 3D viewing in the full colour spectrum. If using projectors for the system, only one is needed. On the other hand the brightness of the image is lost and the frame rate has to be doubled, which in essence, duplicates the hardware requirements of the equipment. Computations and buffer swapping have to be completed much faster, but the refresh rate of the projector or television also has to be twice the speed. This fact restricted the use of screens to the CRT models since it was very expensive to obtain flat screens that could achieve a high enough frame rate for an affordable price and its usage was restricted to investigation and simulation (Turnet & Hellbaum, 1986 [11]). Current home systems for entertainment can handle such a large load (especially for high definition) and it has not been until very recently, that the active systems have become popular and spread very quickly.

2.1.2 Displays for curved projections

It is now time to discuss possibilities that may be used to present the graphic information. Most of the systems use projected screens because it seems to be the solution that offers the greatest displayed area, good resolution and stereographic images with polarized light at a relatively affordable price (depending on the projectors). In some other situations, normal screens are used. If the system does not need to be particularly big and the screen is able to offer 3D (maybe through the use of *shutter glasses* or a lenticular filter), screens may offer a better result (for example, they are not affected as much as projectors by illumination conditions).

There are many types of screens; *Cathodic Ray Tube* (*CRT*), *Liquid Christal Display* (*LCD*), *Plasma* and so forth, but it is not the purpose of this section to compare them. Whichever the type chosen, a large surface is desirable in most situations (larger sizes than what can be offered by a single screen). A curved surface would also be an interesting screen option in order to enhance the realism and level of immersion. For this reason, the highlighted options are curved screens and customable clusters of screens.

2.1.2.1 Cluster of screens

A cluster of screens is composed by a set of displays, set close to each other to increase the displayable area of the system. As the scene is displayed on smaller screens than the whole surface, it can be configured in any shape. For example, screens could be easily located forming a curved surface (sector of a cylinder) to provide a panoramic view in front of the user. The main drawback of theses configurations is that the frames of the screens will interrupt the view giving the displayable area a gridded appearance. In the image below (fig. 11) there is an example of multi-screen display.

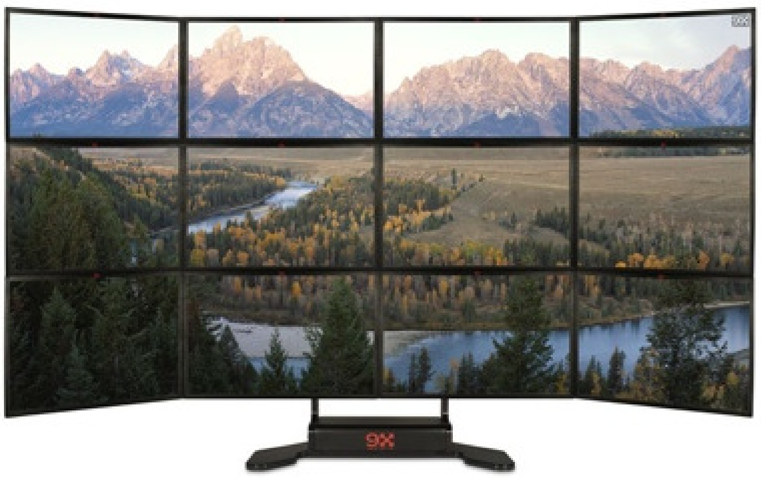


Fig 11: Example configuration of a cluster of screens for a panoramic view.

On the other hand, this configuration is easy to implement and offers a very high definition and brightness. If the stereo is necessary and the monitors offer the possibility of using a *shutter-glasses* based system (that is, they offer a high enough frame rate) it could be an interesting option to take into account.

2.1.2.2 Curved screens

This technology is relatively new and only a few brands have released a very small amount of models compared to the flat screen market (flat in terms of the glass surface). The screen is much wider than normal models but it is the fact that it is curved what makes the experience so much more immersive. The future of these new displays is very promising, since they offer a much more realistic way of presenting ~~the~~ information, especially if the contents shown ~~in~~ on the screen have been calculated for that specific curvature and aspect ratio. At the moment, the price of these devices is still relatively high, and the size can also be an issue, even though the technology that is being in use is *TFT* (*Thin Film Transistor*) *LCD* or *OLED* (*Organic Light-Emitting Diode*).

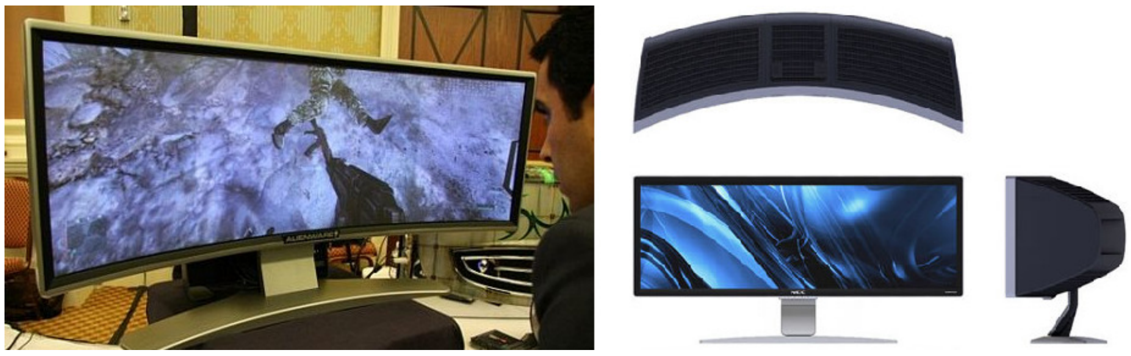


Fig 12: Curved monitor by *Alienware* (left) and *NEC* (right).

2.1.3 Projectors

Another way of displaying information is by means of using projectors. These devices take a video signal and project the corresponding image onto a surface using a lens system. One of the best characteristics of a system using projectors is the fact that the displayed area is highly configurable, and so, it offers a greater flexibility to the systems. The projectors are more sensitive to the lighting conditions than screens are, but on the other hand, they offer greater possibilities for stereo applications due to the fact that is easy to provide the projectors with light polarizer filters and use them in conjunction with polarized glasses.

Issues to be taken into account when using a projector are not only the ones related to the maximum resolution, but also brightness. A projector with a higher light output (measured in lumens) is needed for larger screens or rooms with a high level of ambient light. It is very important to take note that the total amount of light does not change, and consequently as the size of the projected area increases, brightness decreases. There is another factor related to brightness that is also very important; the black level. The black level is the level of brightness at the darkest (black) point of an image and if not properly adjusted, visual information which is supposed to be black would not be completely black, and vice versa. Contrast is another parameter to consider, and adjusts the white level.

There are a lot of different projectors available on the market, and only a brief description of the most important types is given in the sections below, comparing strong and weak points.

2.1.3.1 Cathodic Ray Tube projectors

CRT projectors use a high-brightness tube as the image generating element, which uses a lens placed in front of the canon to enlarge and focus the output image onto a screen. Most of modern CRT projectors have three separate CRT tubes (see fig. 13), one for each colour (red, green and blue) which complicates the process of adjustment to align the images. It is the oldest system still in regular use despite the bulky carcass that is needed.

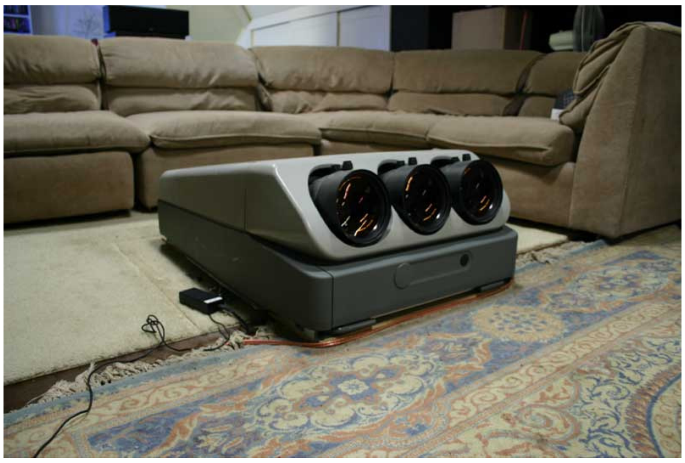


Fig. 13: CRT projector by *Sony*.

The CRT projectors provide the largest screen size with accurate colour reproduction for a given cost (up to 1920x1200, even though few projectors can reach resolutions up to 3200x2560 sacrificing their ability to resolve fine detail). *CRT* also offer superior black level compared to *LCD* and *DLP* (see next sections). As with *CRT* screens, resolution and refresh rate are configurable within some limits. They also offer a very long service life (up to 10.000 hours), maintaining adequate brightness.

2.1.3.2 Liquid Crystal Display projectors

The way LCD projectors work, consists of sending light from a metal halide lamp as they represent a compact, powerful and efficient source of light while maintaining relatively good temperatures and have a broad spectrum colour. The light is passed through a prism that separates it into three poly-silicone panels, one for each component of the video signal. The most important brand developing *LCD* projectors is *3LCD* which was used for the first time in projectors in 1988 (see fig. 14). As polarized light passes through these panels, individual pixels can be opened to allow light to pass through, or closed to block it. This combination of opened and closed pixels is what produces the wide range of colours in the projected image. *LCD* projectors can be thought to be the simplest system when compared to *CRT* and *DLP*. It is the most common and affordable projector used for home theatres and business.

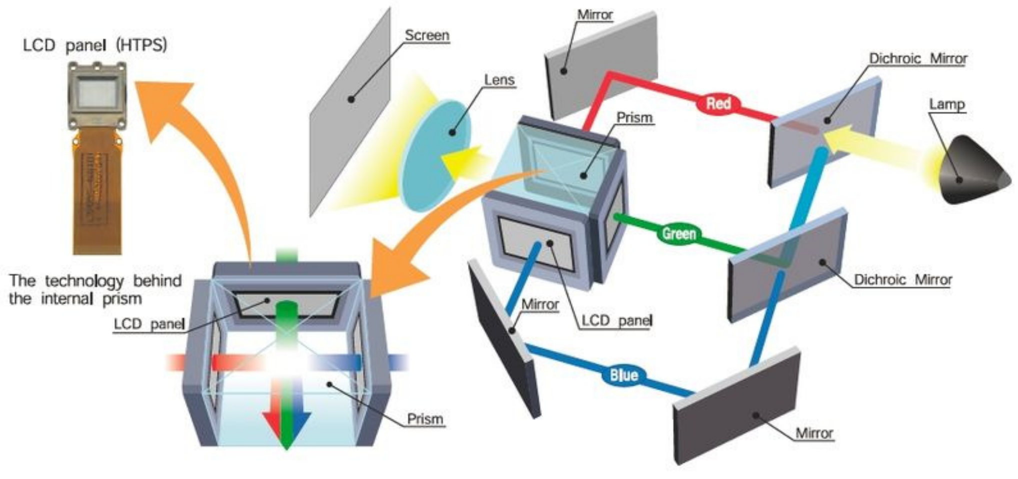


Fig. 14: Intern scheme of components for a projector using *3LCD* technology.

The biggest problems with this technology are the pixelation effect, and poor black level and contrast, (even though the most modern models have improved on these). Nevertheless, they have been overtaken by the DLPs (see next section). An advantage of these projectors is that they are very small in size when compared to the CRTs.

2.1.3.3 Digital Light Processing projectors

This technology was developed by *Texas Instruments* in 1987 and is not only employed in projector units but also rear projection televisions (having supplanted *CRT* rear projectors and are currently competing against *LCD* and Plasma in the high definition television market).

It works quite differently to a *LCD*. In *DLP* projectors, the image is formed by microscopic mirrors placed in a matrix on a chip known as *Digital Micromirror Device* (*DMD*). Each one of these mirrors correspond to, at least, one pixel in the projected image (ideally just one) so it is directly related to the definition of the device. These mirrors are able to wobble back and forth, directing light either into the lens path to turn the pixel on, or away from the lens path to turn it off. Following this idea, there are two major methods; single-chip *DLP* projectors, and three-chip projectors (though there are also other approaches). For the single-chip option, colour is produced either by placing a colour wheel between the white lamp and the DMD or using individual light sources to produce the primary colours. The *DLP* chip is synchronized with the rotating motion of the colour wheel so that the colours are displayed sequentially at a sufficient high rate that the observer sees the whole image (most systems operate at up to 10x the frame rate). The second option, the three-chip projector, is more expensive, and there are three separate *DMD* chips (red, green and blue) so the light from the lamp needs to be split by a prism and routed to the corresponding chip (see fig. 15).

This mechanical spinning colour wheel may exhibit an undesired effect known as '*rainbow effect*' that is perceptible especially when the projected content features high contrast regions moving (i.e. the credits of a movie). The eye follows the moving object with a constant motion but the projector will display each alternating colour of the frame at the same location for the duration of the whole frame. For this reason, the observer will see a frame of a specific colour (for example red) and the next frame ~~it~~ will be different (at the same location overlapping the previous colour). As this is occurring the eye will have moved towards the object´s next

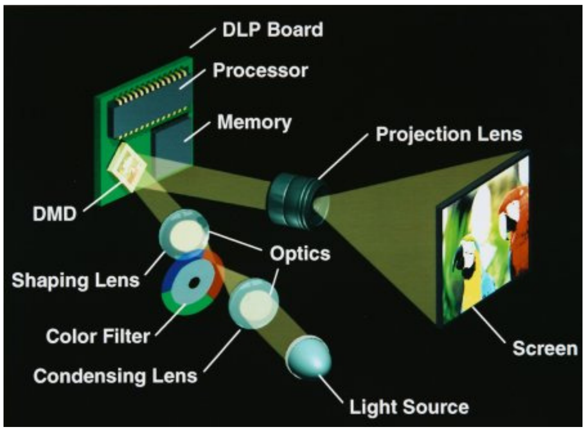


Fig. 15: Single *DLP* chip based scheme.

frame target. The third colour is then displayed and the same problem will occur again. This alternation is what produces the 'rainbow effect', even though some people never notice it. Other *DLP* variants that do not use the spinning wheel, minimize the rainbow effect by using multicolour laser-based and led-based chips, whose pulse rate is not limited by physical motion.

On the other hand, *DLP* projectors tend to be very compact (especially the single-chip models), and they produce smooth, high contrast video exempt of pixelation (unlike the *LCD* projectors).

2.1.3.4 Projected screens

There is not a definitive answer to the question of which technology is best. It depends on the use, since all the projector types have advantages and disadvantages. It is important to understand the main differences between them to be able to select the right technology for every system. However, selecting the right projector is not enough to assure a good quality and has to be chosen in conjunction with the appropriate screen, on which it is to be projecting. Again, depending on the situation, some options are more appropriate than others.

Projection screens might be permanently installed (i.e. movie theatre), mobile (i.e. conference room) or painted on the wall. Normally, white or grey screens are used to avoid discoloration of the image while the brightness depends on multiple factors, such as the ambient light levels and the luminous power of the projector. There is a trade-off between brightness-darkness due to the fact that black colour is not projected (as black is the lack of light) and black areas of the image will be represented by the screen itself. For that reason, one might think that black (or dark) surfaces are more appropriate as screens, but bright areas of the image we would lose all the colour vivacity on a dark surface. For this reason (in attempt to find a balance) screens most commonly used are grey.

Screen surfaces can be categorized in four main groups; the first group is matte white screens, representing a surface with less than 5% reflectivity (black looks very dark grey but images do not look very bright). Another type of screen is pearlescent (obtaining a reflective coating to a surface of matte vinyl) of which reflectivity is typically 15% on which black looks dark grey, while also maintaining image brightness, providing best overall contrast . Silver screens offer 30% reflectivity, black appears medium grey and the image appears very bright which causes the dark colours look a little dull. The last type is glass bead screens. These screens use thousands of tiny glass marbles embedded in a transparent coating on the surface of the screen and provides the highest level of reflectivity (typically around 40%) and consequently, black usually appears light grey and the image is usually too bright (this kind of projector is used only under special conditions).

Another property often quoted about projected screens is gain, which typically varies in the range 0.8 of light grey matte to 2.5 of the reflective glass bead screens. Gain is a measure of reflectivity compared to a screen coated with magnesium carbonate or titanium oxide (bright white colour) when the measurement is taken for light targeted and reflected perpendicular to the screen. Low gain screens (0.9 to 1.3) are highly diffusive, and as they diffuse the light in all directions, no matter where the users are, the image will be equally bright and clear. Greater gains can be achieved by using materials that reflect more of the light parallel to the projection axis and less off-axis. The highest levels are achieved by a mirror, even though the audience would see the reflection of the projector, defeating the purpose of using a screen. Many of the screens that are usually employed are not perfectly reflective surfaces but semi-glossy, and so exhibit mirror-like properties. As they have mirror like properties, a bright ‘hot spot’ on the screen may appear. This brighter area is produced by the reflection of the projector lens, magnified and blurred. Most users do not notice the presence of the brighter spot, mostly due to the fact that humans are apt to noticing contrast in small areas but less so in luminous variations as great as half of the screen. Some screens are retro-reflective, and unlike the reflective (mirror-like) ones they reflect the light back toward the source (see fig. 16). Hot spotting is also less of a problem with high gain retro-reflective screens. Semi-specular high gain screen materials are suitable ~~to~~ for ceiling-mounted projector setups, so that the greatest intensity of light will be reflected downward towards the audience. For the same reason a viewer located to one side of the audience the same area will look darkened (although some screen materials are semi-specularly reflective in the vertical plane while more perfectly diffusely reflective in the horizontal plane to avoid this). Retro-reflective screens (i.e. glass-bead screens) reflect light back to the source more intensely than any other direction so their ideal configuration is when the image source is placed facing the same direction as the audience to the screen. The screen centre may look brighter, but unlike semi-specular screens this does not depend on the viewer´s position. Retro-reflective screens are seen as very desirable due to the high image intensity they can produce with a given luminous flux.

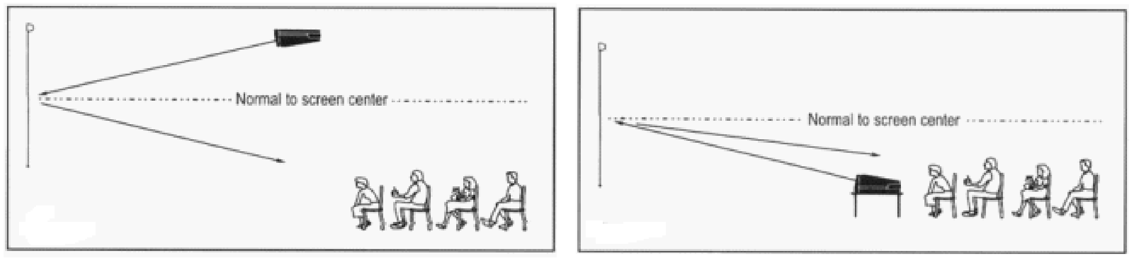


Fig 16: Reflective screen (left) and retro-reflective screen (right). (*Da-Lite Education*)

In addition to the possibilities described above, screens can also be designed to be projected from behind, so they do not reflect light but rather transmit it to the opposite face of the screen from where it is being projected from. The setup of these systems is usually more complicated but has some desirable properties such as avoiding the casting of shadows onto the screen, which is very important in systems where the user can be located very close to the projected area.

The surface of the screen can also be perforated (hundreds of small, evenly spaced holes). Multiple holes can let the air pass through. Using this type of surface, speakers can be placed directly behind the screen which is very desirable for surround sound systems.

The screen can also be positioned into a curve. Curved surfaces can be made highly reflective without introducing any visible 'hot spots', provided the placement of the projector and viewer´s position are arranged properly. If the projected light is directed to the audience and the angle of reflection is the same across the screen, no artifacts are formed and the screen can be though as a large 'hot spot'.

However, sometimes when the area to be projected is relatively large compared to the volume covered by the projector, there are additional problems to take into account. Brightness can be much higher in the area located directly in the front of the light cannon while much darker in the surrounding area, fading out proportionally with distance. The use of more than one projector can also be of interesting, in order to attempt to maintain a wider projected area with uniform brightness. In order to do this, it is necessary to deal with other problems that will be discussed in the next section and the Analysis chapter.

2.1.3.5 Warping/Blending and software

When projectors are located at an awkward angle (very oblique with respect to the screen), the surface is not flat but curved, or if using two projectors for the same area (for example when we display stereographic images) is necessary to make a geometric correction ~~of~~ to the images projected. Most of the projectors offer the possibility of using *Keystone* correction (the *Keystone* effect occurs when trying to project an image onto a surface at an angle, so the image ends up looking like a trapezoid and is unfocused in some areas because lenses only focus for an average distance) but very few offer curved corrections.

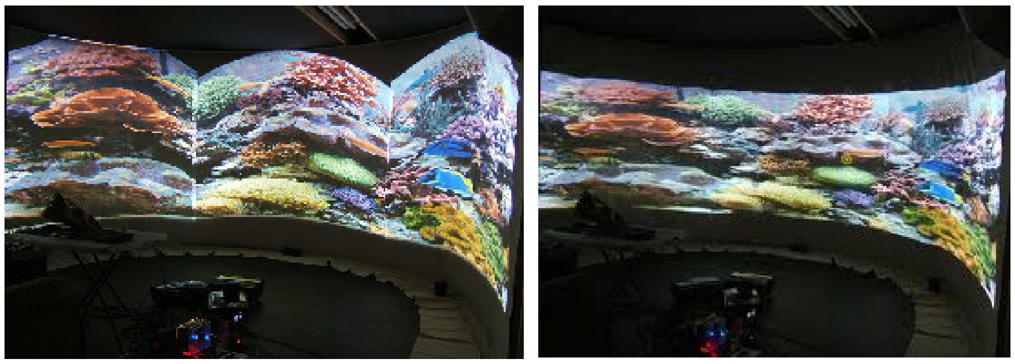


Fig 17: Three projectors without warping or blending on the left (a) and projections corrected (warped and blended) on the right (b). Done with *Sol7* (*Immersaview*).

Another process that the images may need to undergo is blending. In order to achieve a greater projectable area, multiple projectors can be arranged in such a way that they have an overlapping region between them with coincident geometries o the transition is smooth. However, by doing this, the overlapping region shows brighter colours because of the additive nature of the light (there is light from both projectors in this region). The solution is to blend this area. For doing so, the image projected must be faded smoothly so the brightness level eventually fades to black.

Warping and blending are processes that very often need to be performed together. The details can be found in the next chapter, for the moment some existing software that can be used for these purposes. This software is presented below, as well as a warping method that calibrates the projectors automatically

2.1.3.5.1 Existing calibration programs

One possibility for calibrating a system consists in generating one or more configuration files with all the necessary information to warp and blend the output images in order to adapt them to the screen. For instance, *iCinema* centre (see section 2.2.3) developed *AVIEconfig*, a calibration system specialised for *AVIE* (see section 2.2.3.1.2) that outputs the necessary files with the calibration data, defining the warping meshes and blending areas. This data can be parsed by any application to obtain the necessary information to perform corrections. Another possibility is to have the calibration program working at *driver* level, which means that anything that is represented in the final raster of the graphics card will be affected by the corrections. An example program that works in this way is *Sol7*.

Sol7 is a software tool by *Immersaview* for geometric correction (warping) and edge blending of projections for non real-time and real-time applications. It was especially designed for curved screen displays, but also aids the projector alignment of non-planar and off-axis displays. The way *Sol7* works is very intuitive for both warping and blending, offering a very straightforward integration between the two processes.

The warping can be linear of curved (depending on the kind of screen to project). Firstly *Sol7* displays a grid without any warp correction applied. The user can add or remove control points so the adaptation to the surface is easier. When the calibration is done, the grid appears aligned to the projected screen, even though the output from the graphics card might appear twisted. The correction being applied is defined by the warping mesh that the user has defined by modifying the location of the control points. In fig. 18 there is an example of a basic mesh without any warping applied (a), a linear correction (b) and a cylindrical warping (c). The images to be displayed will be affected in the same way that the grid texture is displayed. The areas in the background outside of the grid are black because the projector does not have to project any light in those regions.

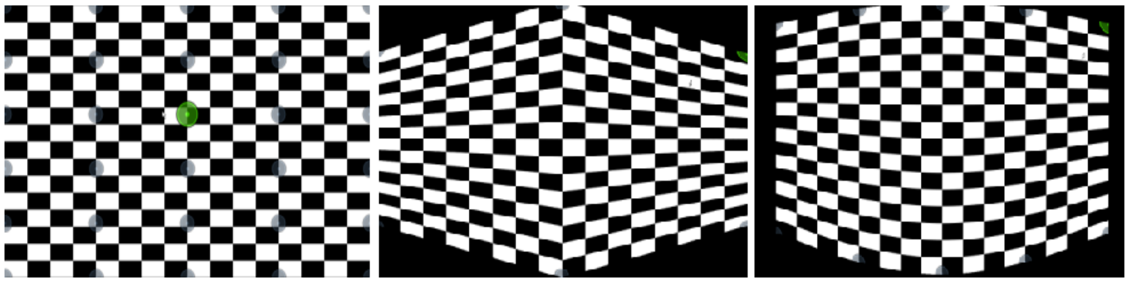


Fig 18: *Sol 7* mesh without any warp on the left (a), linearly corrected in the middle (b) and cylindrically corrected on the right (c). (*Immersaview*)

If we are using more than one projector, normally the warping is applied in a way that there is a well defined overlapping region. For simplicity, this area will be a specific number of squares (for example 2). It does not necessarily need to be done in this manner, but in any case the application that calculates what it is shown in each warped mesh has to be consistent with the overlap. As it has been mentioned before, overlaps are necessary to achieve a smooth transition between the different projected areas (normally associated with different projectors), but when there are two or more projectors displaying contents in the same area, brightness is higher than it should be. *Sol7* allows the user to switch between warping and blending mode, so this correction can be done easily as well. Moving the edge where the blending region has to finish and modifying the parameter that defines the blending function users are able to ~~get~~ obtain a smooth transition between projector areas (see fig. 19). Later versions of *Sol7* allow the control of the blending region with more than one control point.

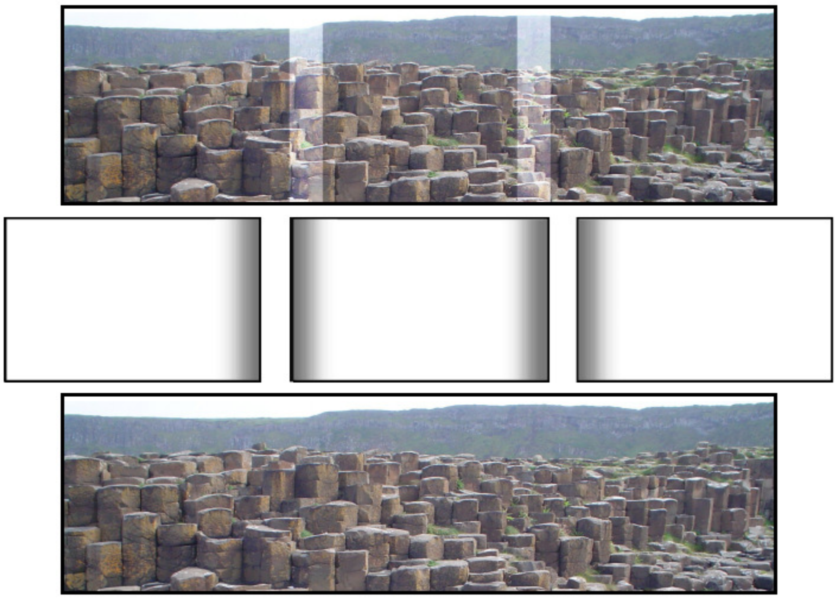


Fig. 19: Image without blending correction at the top, blending masks in the middle and final image with the masks applied at the bottom (*Immersaview*).

For this thesis, a calibration program that incorporates warping and blending has been implemented. Some of its features have been specially designed to improve the process of calibration for an *AVIE* system, even though it can be used for many other systems. It also incorporates some features that improve those contained in *Sol7* like the possibility of controlling the tension of the curves that define the surface (a wide range of curvatures between linear and highly curved are possible to achieve). It is possible to change the definition of the mesh as well, giving greater flexibility for the blending configuration. These and other features will be presented in the next chapters.

2.1.3.5.2 Automatic projector calibration

Calibration process can sometimes be tedious. In particular if it needs to be repeated often because the system is mobile or because the screen and projector are prone to vibrations or movements that would cause inaccuracies in the calibration. For this reason, automatic projector calibration has been previously explored, with two different approximations. The first, consists of using camera-based computer vision techniques, for example Raksar, 2001 [12] and Sukthankar, 2001 [13]. Both of these approximations differ from the way the system elements are located, for example in [12] the camera and projector are attached to each other and point in the same direction (an additional tilting sensor is also located in the projector so a global vertical direction can be established) while in [13] the camera can be located anywhere as long as the screen on which the image is to be projected is within the field of view of the camera. The fundamental idea to perform the correction automatically is to use using homographic transformations [14]. In computer vision, any two images of the same planar surface are related by a homographic relationship (assuming pinhole camera models), which in this case, allows us to calculate the relative positions from the camera to the projector using the screen as the common plane and correct the projected image under the point of view of the camera.

Calibration techniques based in the use of cameras has been extended to also be able to work in curved screens like in [15] (Lee, 2009), [16] (Baar, 2003) or [17] (Sun, 2008), even though precision with general curved surfaces is still an issue. Calibrating automatically onto non-planar surfaces is complex, has high computational requirements and the camera-projector geometry must me precisely known and remain static.

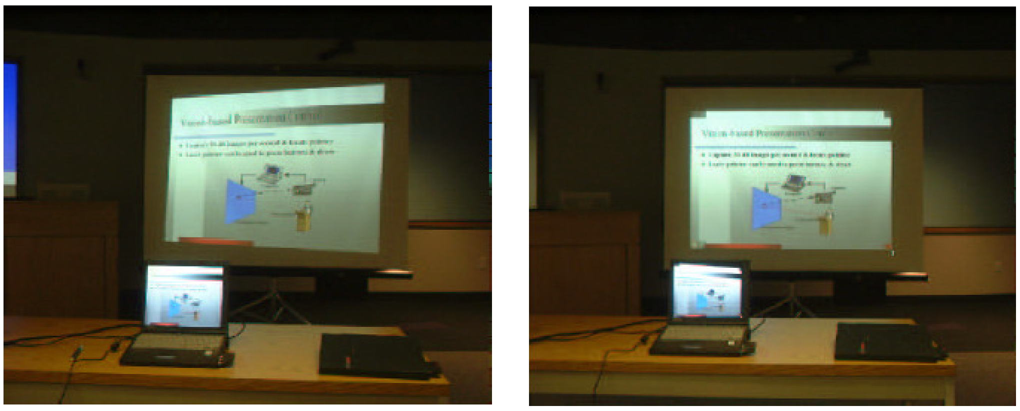


Fig 20: Image projected onto a screen without the correction applied (left) and applied (right). (Sukthankar, 2001 [13]).

A different approach for doing this calibration without using cameras is one proposed in Lee, 2004 [18]. The idea is different in this case; it consists of calibrating onto existing physical features such as a projection screen frame. The features are determined by embedded optical sensors in the target surface at the points of interest. By doing this, problems related with the camera resolution, reflexive properties of the target surface, lighting conditions and background separation are eradicated, obtaining an accurate and robust result, while keeping the problem simple. Once the surface to project on is located within the field of view of the projector, the system starts to display several black and white patterns, and receives the feedback from the light sensors (they will inform when a flux of light is affecting them). Doing this with the different patterns, it is possible to determine which are the specific pixels of the image that will be illuminating each one of the receptors and consequently, define homographic transformation. This method could be extended to work on curved surfaces if enough light sensors are located defining the shape of the curved surface.

2.2. Examples of immersive systems

There have been some immersive systems that have served as precedents and examples of design paradigms. Two of the most influential ones in the industry are discussed below.

2.2.1. *Cave Automatic Virtual Environment*

The technology *CAVE* (*Cave Automatic Virtual Environment*) is an immersive virtual reality system that was proposed in 1992 (Cruz-Neira, 1992/93 [19] [20]). It uses a compendium of techniques of tracking, immersive visualization, stereographic projections, and so forth giving support ones to each others.

A *CAVE* consists of a cubic room whose flat walls are retro-projected displaying the virtual environment. Floor plane is normally projected from above, but ideally the floor and ceiling planes would be projected from behind similarly to the walls. The projectors employed for this purpose are high definition *CRT*s (two for each wall, which implies 6 light cannons). The user enters the room wearing shutter glasses (explained in previous sections) that determine the exact position and orientation of the head, which is done electromagnetically or optically. The synchronization of the glasses with the system is done by infrared, Bluetooth or some other wireless technology.

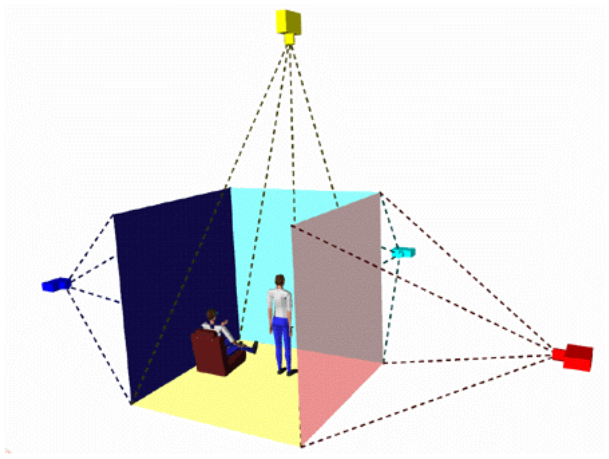


Fig 21: Scheme of the location of the projectors of a *CAVE*.

With all of the previously mentioned date/information, together with a surrounding 3D audio system and possibly haptic devices, the user has the impression that they are immersed in a virtual world and can walk around (and even within) the objects freely. The perspectives calculated are determined by the planes of the walls and the eyes of the user. For that reason, the point of view will only be correct for one user at a time.

The fact that the projection surfaces are planes can have some undesired effects. First of all, the corners are undesirable for the visualisation. Even though the projections are calculated

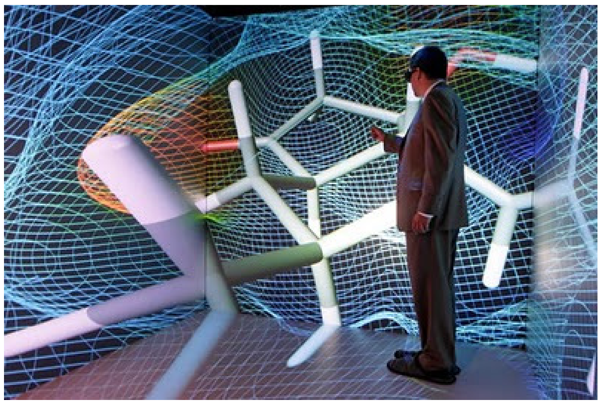


Fig 22: User interacting with the virtual environment in a *CAVE*.

properly for the user, there is not a seamless surface in this system, as the corners very obvious. Also, the scene that is displayed to the user is normally a panoramic view, but projected onto flat surfaces, which is probably not most ideal, as it will be explained in the next chapter.

CAVE systems are extremely expensive which constrains the application fields in which they can be used. Only large companies and institutions can afford them. By using them. the design phase and prototyping of products becomes notably sped up. The system is also very useful for investigation purposes, exploration, engineering and so on.

2.2.2. Immersive rear projection on curved screens

This project (Kolb, 2009 [21]) is very recent and interesting to mention because is one of the very few curved retro-projected screen existing immersive systems. It is a Virtual Relity (VR) installation at the University of Siegen (Germany) that consists of a 180º cylindrical rear projection screen and a front-projection floor. It is being used for both immersive VR applications with user tracking and for performing presentations for an audience.

It consists of 6 projectors (4 behind the screen and two at the top of it), a semi-cylindrical screen with a diameter of 2.5 meters and 2.6 meters high. The 3D is supported by *Infitec* technology. The tracking is done using infrared sensors located on the ceiling.

In the picture below (fig. 23) the system is being used for giving a conference to an audience wearing the *Infitec* glasses.

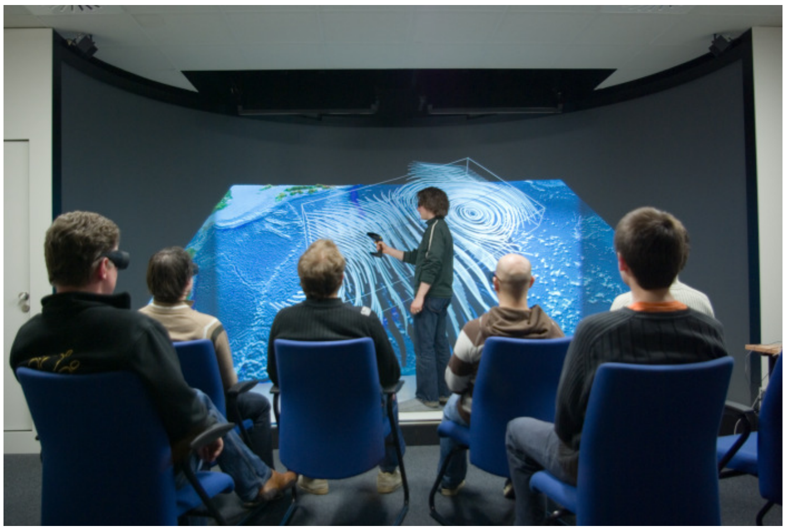


Fig 23: Rear projected curved screen in Siegen University (Germany).

There are many other immersive systems and, because it is the purpose of this thesis, special emphasis has to be placed on those involving curved projection spaces. More examples are presented in the next section, all of them developed by, or in collaboration with the *iCinema* centre at The University of New South Wales.

2.2.3. *iCinema* Research Centre

The *iCinema* Centre for Interactive Cinema Research has been the background of the work developed in the present thesis since accepting me during the development of this masters' thesis. *iCinema* was established in 2002, being a joint venture of the College of Fine Arts and Faculty of Engineering at the University of New South Wales. It brings together researchers in new media, aesthetics, cinematic theory, multimedia design, computer science, cognitive science, software/hardware engineering and mining virtual reality. The *iCinema* research program focuses on interactive systems applied to arts, culture and industry and the way digital contents can be used to redefine recreation, learning and the way we work and do business.

The research areas cover fields such as immersive visualization systems, interactive narrative systems and distributed interfaces. These areas are interconnected and are based around the main idea of immersing users into a multimedia world, giving importance to other aspects aside from focusing solely on the technological aspects of displaying three dimensional contents in a realistic way. Very often the contents have to support the possibility being accessed and edited as well as view multi-temporal media simultaneously, providing the basis for greater experiential and imaginative immersion for the viewer in their interaction with multiple narrative fields.

This transcends conventional literary and cinematic theory in which the interaction is linear (uniqueness of digital narrative). The interface used has to offer a seamless integration between the contents and the user input, providing a greater multi-sensorial responsiveness. The resolution of all this research has applicability in the fields of, amongst others, cinema, interactive video, multi-media presentations, defense training, medical visualization, architectural planning, multi-media games, on-line research, education, and so forth. In particular, the topic of this thesis is framed within the immersive visualization area.

2.2.3.1 *iCinema´*s most significant platforms

In this section three of the most representative platforms developed by *iCinema* are presented, for both capturing and displaying contents. They are extremely representative of the state of the art regarding immersive panoramic systems (Kuchelmeister, 2009 [22]).

2.2.3.1.1 *iDome*

iDome is a proprietary hardware/software platform developed by the *iCinema* Centre. It offers a cost effective and compact immersive visualization environment for panoramic or spherical projections. Contents displayed can be pre-recorded (video) or computer generated (real time or not).

The projection system used by the *iDome* is formed by a semi-spherical fibre-glass screen three to five meters in diameter that stands vertically in the front of the user, and a projector that is able to project images onto a surface of 180º that covers the peripheral vision of the user and thus results in a truly immersive experience.

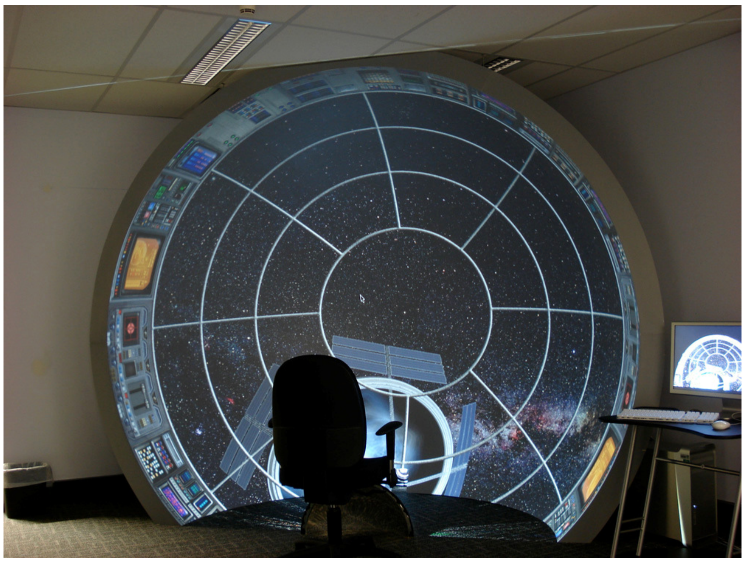


Fig 24: ***iDome***

To cover the whole surface in a system like the *iDome* several possibilities are available. For example, more than one projector could be used and the images projected by each of them can be warped and blended properly. However this is very restrictive spatially speaking because the user stands in the front of the screen and shadows could be casted onto the screen. In addition, the configuration of the projectors would be more problematic. Another possibility would be to use a fish-eye lens for the projector so it spreads the light all over the screen, which requires a stand for the projector or for it to be located in the middle of the screen which blocks the view partially (see fig. 25a).

iDome utilizes one single projector (avoiding synchronization and blending issues) and a spherical mirror as a reflection surface to be able to cover the whole projectable surface (Bourke, 2005 [23]). The projector faces the opposite direction to the screen and projects the images onto the mirror in the lower part of the screen, more or less where the user stands (see fig. 25b). This approach takes better advantage of the space and is much more cost-effective. It also offers the possibility of changing the projector very easily with one with higher resolution when it becomes available (or affordable), which is very important for this kind of projection as the pixels that are projected onto the mirror are reflected to the screen, they may look quite big. It also happens that the output image has to be warped to fit the screen with the consequent lose of resolution.

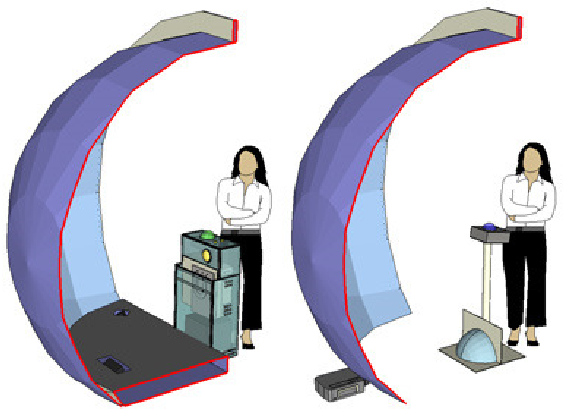


Fig 25: Fish-eye configuration on the left (a) and Spherical mirror configuration on the right (b) (Bourke, 2005 [23]). (*iCinema*)

Bourke also proposed in [24] an extension of the *iDome*, called *iSphere* that has not yet been implemented but would provide a 360º spherical display surface horizontally, by 120º vertically. It also uses a spherical mirror and 4 projectors located at the top of the sphere.

For these kinds of screens, in case the scene displayed is synthetic (generated by computer) the projection has to be spherical in order to obtain a correct and convincing view for the user. Otherwise, if a video is to be displayed, then it has to be acquired properly. This means that it has to be a panoramic video with very wide angles, recorded as cylindrical, spherical or cubic maps since conversions between them are possible. (Bourke, 2002/2010 [25][26]). A normal camera only records perspective videos and is not suitable for such a purpose, so it is necessary to use special devices for that goal. One example is the *iCinema´s Spherecam*.

2.2.3.1.2 *Spherecam*

As explained above, when it is necessary to display panoramic/surrounding images they have to be captured in the appropriate format. Global recording technology works beyond perspectival framing constraints of traditional lenses and it constitutes an expanded model of representation that is closer to the way human beings actually perceive the world. Two different approaches are possible; static image and video capturing.

The idea of capturing 360º panoramic images (this is, normally for cylindrical projections) consists in setting a pair of cameras displaced horizontally, one in respect to the other, at a determinate distance (interocular distance). By rotating the camera pair around the vertical axis by small angle increments while they take multiple shots it is possible to obtain a set of regular pictures to form the panoramic image. If the most accurate result is wanted, the aspect ratio of the captures has to be small (images must have very reduced width), otherwise the deformation introduced by the perspective projections would complicate the post processing of the multiple images to compose them into a single one. This will be discussed in the next chapter. However, if it is necessary to record a sequence of panoramic images (panoramic video) having one or two cameras rotating around a vertical axis is no longer valid since every frame needs to be obtained in the same instant. For doing this, it is not possible to use a single normal lens, since it would need to capture 360º simultaneously. The most exploited method consists in setting a cluster of cameras pointing in different directions. It is possible to configure the cluster of cameras in such a way that there are lenses pointing in many directions around the vertical axis, while also locating cameras pointing to higher or lower orientations in order to achieve spherical projections. Stereo video is complicated to obtain, since twice the amount of cameras are necessary and it is difficult to set them in the same cluster.

iCinema´s *Spherecam* enables numerous camera configurations, depending on the desired vertical field of view, image resolution and presentation format (cylinder, dome, and so on). In the fig. 26 *Spherecam* is configured with 8 cameras placed around the vertical axis, plus an additional fish-eye lens pointing upwards (see fig. 26) providing a view volume of 360º by 240º.

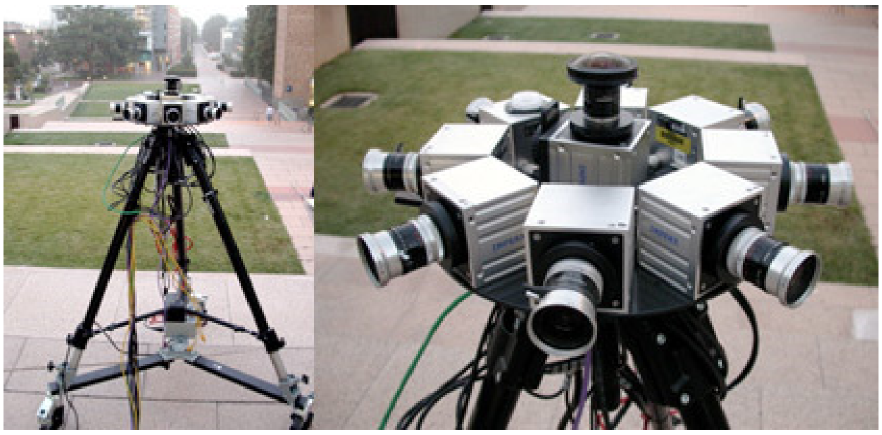


Fig 26: ***iCinema´s Spherecam.***

The full system is composed by the cameras, mounting plates, tripod, recording racks and a twelve channel microphone to record 3D audio. *Spherecam´s* maximum total resolution is 24 Megapixel for panoramic/spherical recording. The raw uncompressed data from the cameras is streamed through capture boards directly onto hard disk arrays within the computers located in the recording racks. To be able to compose the different videos into a unique seamless one the system has to be calibrated. In calibration, the camera´s intrinsic parameters (focal length, lens distortion, centre of distortion, and aspect ratio among others) and extrinsic parameters (position and orientation relative to a global origin) have to be known in order to be able to distribute, distort and blend them into a single one. The option of a last step is to compress the data to make it more compact. *Spherecam* has been used in multiple *iCinema* projects to capture contents for *iDome* and *AVIE*. There are also some studies that have been conducted regarding real-time warps for panoramic images in real-time (Liu, 2002 [27]).

2.2.3.1.3 *Advanced Visualization and Interaction Environment* (*AVIE*)

iCinema´s *Advanced Visualization and Interaction Environment*, codenamed *AVIE* (McGinity, 2007 [28]) serves as the principal platform for experiments in interactive and emergent narrative, artificial intelligence, human-computer interfaces, virtual heritage, panoramic video and real-time computer graphics, as well as a primary platform for public exhibition of *iCinema*´s projects.

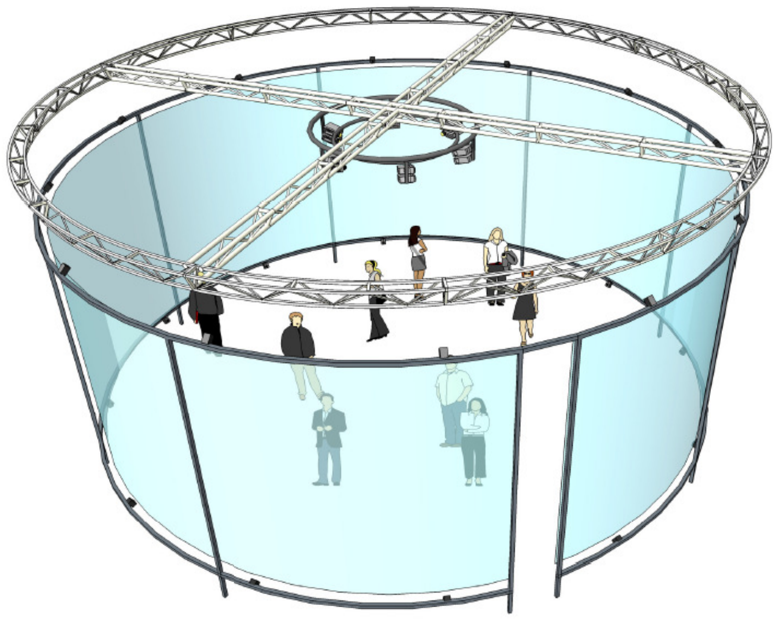


Fig 27: *AVIE* basic scheme.

It consists of a 360 degree cylindrical silvered screen of up to 4 meters in height and 10 meters in diameter on whose internal surface omni-directional stereo panoramic images can be projected, surround audio and marker-less motion tracking providing a highly immersive and interactive environment for up to 20 users in its area of 120 square meters.

A narrow 80 centimetre doorway provides entry to the theatre, offering a vertical field of view of 40º degrees for a user located at its centre. The projectable area is divided into 6 sectors of approximately 60º projected by two *DLP* projectors of 1400 by 1050 pixels, which implies a total of 12 projectors (6 for left eye and 6 for the right) providing a total circumferential resolution of around 7500 pixels. This provides a high pixel resolution, especially for users located in the centre. The silver screen was chosen firstly for its ability to preserve polarization of the projected light (using vertical/horizontal/circular filters for the projector lenses). The images of each 60º region are the output of each one of the six computers *Dual Xeon® Windows ® PC* (connected to the projectors by *DVI* cables). These machines are arranged in a cluster of seven machines, one of which works as a server and synchronizes the projectors. This network is necessary since *genlock* [29] (a technique where the video output of one source, or a specific reference signal, is used to synchronise other image sources together) is usually not sufficient (see section 3.3.6). The master computer also communicates with the audio and tracking system as well as other peripheral devices (such as the console located in the centre of the screen with the joystick, buttons and an orientation sensor).

The audio system is composed by 24 speakers, distributed evenly around the top and bottom of the screen (but they can also be right behind it if the screen if the surface is perforated), providing real-time spatial audio. This feature allows the system to create a fully immersive 360º placement of sound anywhere around the viewers. The tracking system is composed for twelve infra-red cameras and 20 infra-red flood lights, distributed all over the ceiling, giving coverage to the whole theatre and are able to track up to thirty users simultaneously. They are also able to capture the video which is the input source for the tracking algorithms. One of them constructs a *voxel* representation of people within the theatre that provides information about location of head and limbs of the users (Penny, 1999 [30]). Another system tracks in a very accurate way features as fingers pointing in different directions evaluating the local maxima of curvature of the silhouette contours. When more accurate control is needed, orientation sensors can be used as well.

AVIE system has demonstrated during the past few years to be highly portable (*iCinema* members can install an *AVIE* in one day), and allows new interactive content to be imported easily. Proof of that are the several *AVIE* systems around Australia and also Hong Kong, Germany and so forth. Temporary placements have also been set, as in *Seville´s biennale* in *Granada´s Alhambra* (Spain) in 2008.

2.2.3.2 *iCinema´*s example applications

To illustrate how the *iCinema* systems presented can be used in practice, a few examples are discussed below.

2.2.3.2.1 *Historical Darling River Journey*

This application was developed for an exhibition centre and runs in the *iDome* featuring a customized interface. The high-definition video displayed was obtained with a *Spherecam*, using a spherical setup, and for that reason for every frame of the video, a spherical capture of what it was happening around the camera is taken. It was filmed from a boat while it was travelling around a river, and the user can freely control their gaze using the interface provided. If the user is facing a direction where there is an extra narrative layer, or superimposed pictures the attention can be focused on those and jump to them. Doing so, the video becomes not only highly immersive, but also interactive and the information the user gets depends on their choices.

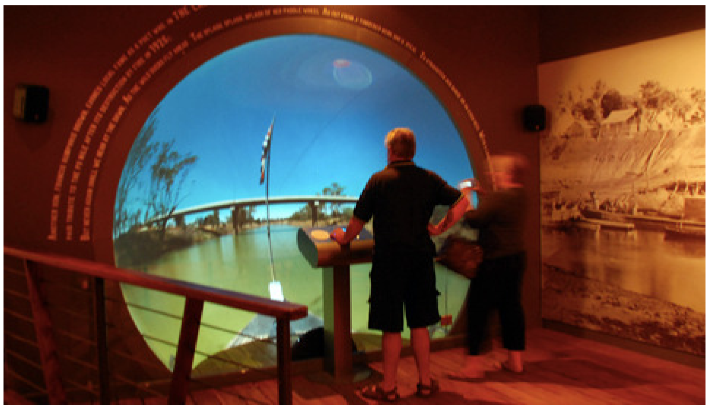


Fig 28: User interacting controlling the camera´s direction as if he was really on the top of the boat.

2.2.3.2.2 *T_Visionarium II*

T_Visionarium II (Del Favero, 2007 [31] & Bennet, 2007 [32]) is the newer version for *AVIE* of an earlier project called *T_Visionarium* (Del Favero, 2005 [33]). *T_Visionarium* projects allow viewers to explore and edit multitude of stories simultaneously, in three dimensions on a 360 degree surrounding screen, in contrast to conventional cinema where viewers passively watch a singular linear story on a flat screen. Users can wander at will through the projection space interacting with the projected information.

For this project, 28 hours of video were segmented into 20.000 video clips (over three hundred displayed simultaneously) belonging to a large media database and displayed in a 3D environment around the user (see fig. 29a and 29b). Each video clip has been tagged independently with descriptors known as metadata defining some properties. Having the video data stored in that way deconstructs the linear narrative into components that become building blocks that the viewer can associate and re-assemble in multiples ways. A very visual way of seeing this would be to select a video, maybe with the orientation sensor or by pointing at it and defining it as a video the user likes. The user could ask to the system for similar videos and it would be possible to regroup them in the 360 degree space in a way the videos most likely to be liked by the user would be in the front of him and the rest behind.

This new way of conceiving and displaying media, in a multi-temporal fashion rather than the traditional linearly of the current cinema [31], opens a new door to a lot of different fields that could become important in interactive cinema, such as data mining, artificial intelligence, recommendation systems, and so forth.

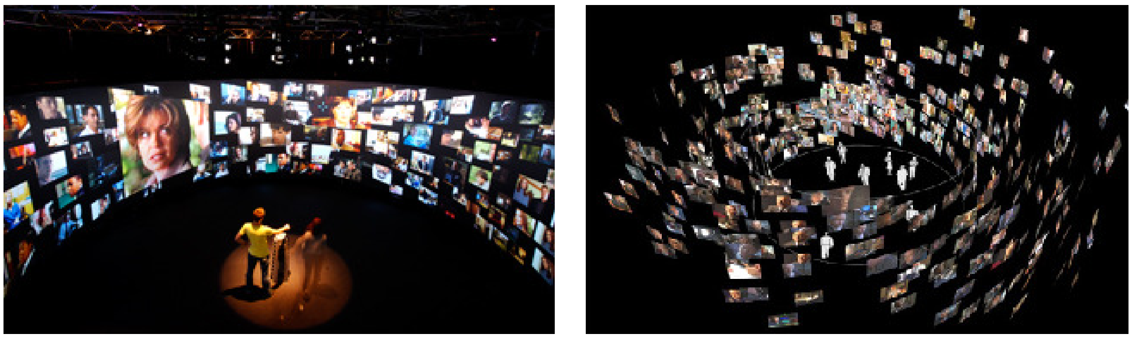


Fig 29: *T_Visionarium II* in *AVIE* on the left (a) and concept visualization space on the right (b).

2.2.3.2.3 *iCasts*

iCasts stands for "*the iCinema Advanced Safety and Training Simulators*" and is the result of an inter-disciplinary collaboration at the University of New South Wales between the

College of Fine Arts, the Faculty of Engineering (School of Computer Science) and the "*School of Mining Engineering*". This project has the ultimate purpose of training Australian miners in a safe environment and "implies" a multi-million dollar agreement to supply virtual reality technologies to *Mines Rescues Ptl. Ltd.* (an important company that provides underground incident response). Importance of such a project is better understood when analysed against how important the primary industry of mining in Australia is and how much it contributes to the national economy.

One of the modules runs in the *iDome* system and provides a 360º by 180º immersive spherical environment for a single user (see fig. 30a). The other module was developed for the *AVIE*, and it supports a group of people training simultaneously (see fig. 30b). In this module, the interaction device (which consists of a console with a joystick and a 3D inertia hand held wand) allow the group of users to move around the 3D virtual world with a high level of realism due to the high definition stereo projections, and existing mine environments modelled that the trainees are able to recognise. The database is being updated continuously, adding features and new locations in order to improve the quality of the training. In fact, modules can run in either *VR* environment (*iDome* and *AVIE*) with the same content, since dynamic switching between different screen projection surfaces and audio systems is supported.

Advantages of such a simulation and training system are obvious. For example, it can be used anywhere at any time, it reduces training impact on production levels, reduces costs and training times, as well as reduces the risks of injuries and damage of expensive equipment.

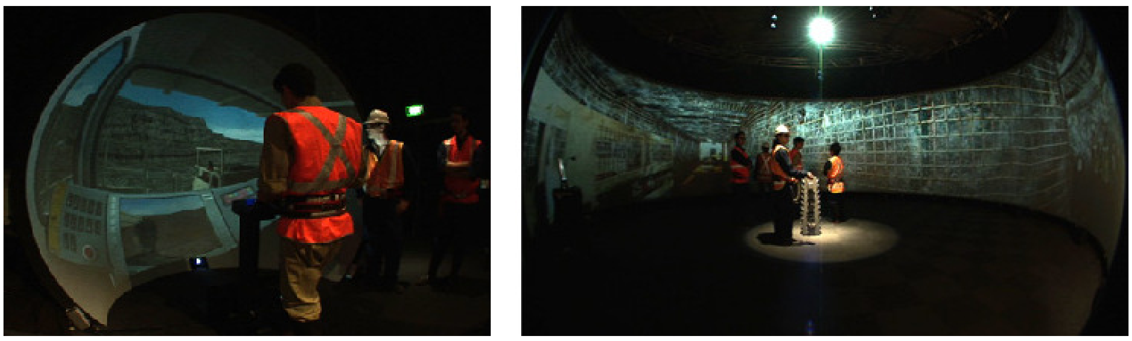


Fig 30: *iCasts* project. *iDome* module on the left (a) and *AVIE* module on the right (b).

iCinema won the *gold IDEA* (*International Design Excellence Awards*) award for *iCasts* in 2009, which recognizes *iCinema´*s research in interaction design, virtual reality and safety training.

Chapter 3

Analysis

The natural way of displaying high wide angle scenes is onto curved screens and this chapter will justify this assumption. However, large curved screens are very rare and expensive and for this reason when a large immersive system needs to be implemented the most common and flexible option usually consists in using projectors. In most cases more than one projector is needed to represent curvilinear spaces onto curved screens. In addition, surfaces at a considerable angle relative to the projector, located at great distances have the problem of a having projections of a very heterogeneous brightness (brighter areas appear closer to the projector´s cannon). On the other hand, in most cases it is impossible to project surrounding panoramic images with a single projector since they normally only project in one direction with a restricted angle. Also, much higher resolutions than those provided by a single projector are usually required.

This implies a new set of problems that have to be solved. Problems like warping, blending, how to calculate high wide view volumes and synchronization are to be analyzed in this chapter. Even though most of the concepts used are common to most of systems which employ spherical or cylindrical projections, special emphasis will be made in cylindrical projections since most of the work developed during this thesis was for the *AVIE* system.

3.1 Warping

Warping can be seen as the generalization of the *Keystone* correction [34]. *Keystone* deformation happens when attempting to project an image onto a surface at an angle; that is the projector is not quite centred on the screen it is projecting on. Consequently, projections that are not facing the screen perpendicularly will project a distorted version of the image, making it look trapezoidal. *Keystone* correction can be performed by most projectors digitally (rather than optically) using internal *LCD* panels or *DLP* mirrors (depending on the technology used), but very rarely the projector offers the possibility of performing more complex corrections like the ones needed when projecting onto curved surfaces, (most of the models do not even have the ability to correct horizontal *keystone* distortions). Even though the quality of the projection is reduced, due to the fact that the number of individual pixels used is reduced by the warping, correcting the image is a process that cannot be avoided in most cases.

The basic idea to conduct a geometric correction is to project a pattern onto the screen and warp it until it appears correct. Normally, this pattern consists of a grid of squares which have to have the same dimension at the end of the process. This correction is done in screen space, modifying the control points that define the grid. When the image is adapted to the surface, what has been done to the output image has been to deform it in such a way that there is a function to map between the output and the projection coordinates. In the image below (see fig. 31) there is a sketch of how a projector would need to pre-warp the image before projecting it, if located above a cylindrical screen (i.e. *AVIE*).

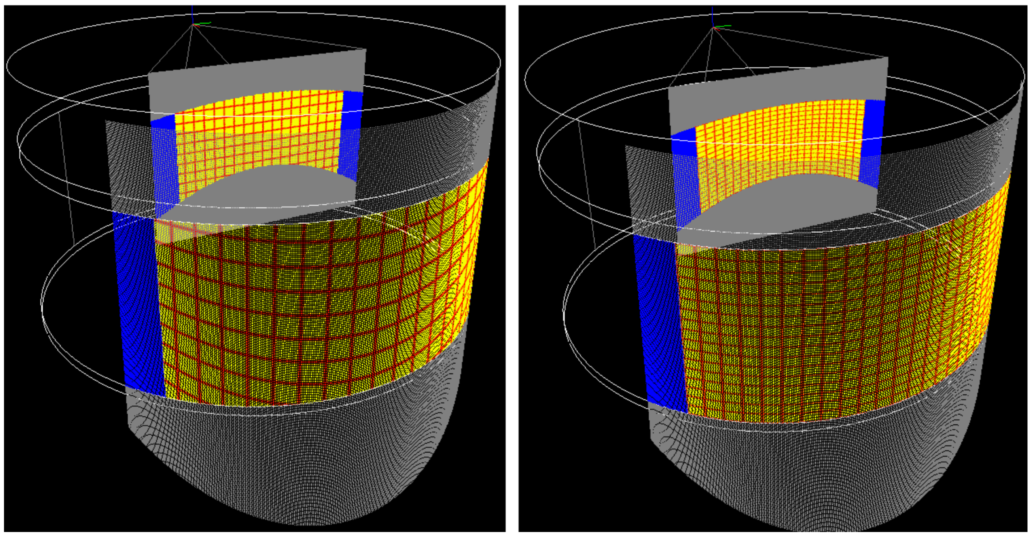


Fig 31: Grid projected onto a cylindrical screen. Original on the left (a) and warped on the right (b). (Bourke, 2004 [1030])

In the left image (fig. 31a) is the output image from the projector has the squares well aligned (same width and length) and all the lines are horizontal and vertical. However, that property is not conserved in the projection onto the cylindrical surface. Squares do not fit on the screen and vary in size, and lines look curved. Once it is warped (fig. 31b) squares look deformed (different sizes and curved lines) in the output image but correct on the projection (all of them have the same size and lines are vertical and horizontal).

It is important to see that warping is only a process that maps between pixels of the projector´s output image and the pixels of the projected image. This means that warping on its own does not convert a perspective projection (like any movie or computer game) into a curved cylindrical projection. The contents have to be calculated in consequence with the appropriate projective method for the screen on which is to be projected. After warping, the output image will look correct on the surface.

Consequently, the method reduces to searching for the appropriate warping mesh. If we have all the data, in theory it is possible to calculate the mesh mathematically. In fact fig. 31 shows two captures of a program that does exactly this [35]. The problem is that in practice it is impossible (or extremely difficult) to know all the exact parameters of the system (projected angle, distance between projector and screen, and so forth). It also may occur that the surface to be projected on has imperfections or the shape has a complicated mathematical definition (it can be a very irregular). For this reason, warping has to be performed manually or with an accurate automatic method that receives direct feedback from the screen projected, either from cameras [16][17] or any other method. A method based on camera feedback to adapt the warp mesh to the screen can be very difficult to utilise and inaccurate, especially if the surface is very irregular. An interesting option that could be considered for this purpose is using embedded light sensors as in [18]. Locating fibre sensors all around the screen and projecting light patterns (just black and white images) could determine which pixel of the projector output corresponds to the sensor on the screen. This information could be used to modify the control points and adapt the mesh in consequence. A simpler approach consists on calculating an approximate version of the mesh (either with a mathematical expression or an automatic method) to tune it manually afterwards.

In any case, what is important is to adapt the mesh as accurately as possible while keeping the process simple, especially if it is going to be performed manually. It is also interesting to have several kinds of curves for the meshes. The warping method implemented allows the user to choose the tension of the curves so the mesh can be configured from linear to highly curved. The possibilities considered have been extracted from [36] (Salomon, 2006).

The mesh also has to have the following properties to make it appropriate for the task. It has to allow the user to control the position of a few control points, with the ability to keep the continuity of the curves but also allowing local control (which discards regular polynomial curves). This means that when one point is modified the mesh is affected in the adjacent areas, but the rest of the mesh remains unaffected. One of the first options considered was to use *Bezier* curves because they offer local control and continuity $C1$ (curves are joined and share first derivative in joints) but was discarded due to the fact that this kind of curve does not pass through all the control points but only the beginning and end points (ibid, pp. 174). Introducing a precise location for a point and forcing the curve to pass through it, while keeping the continuity, would significantly aid in future improvements such as automating the process. Even if that is not the case, keeping the curves controlled, constraining them to pass through the specified points allows better and faster results when adapting the mesh to arbitrary surfaces.

The first prototype of the warping program was developed using bicubic *Hermite* curves structured in a grid of two-dimensional patches. Unlike *Bezier* curves (or *splines*), *Hermite* curves pass through all the control points that define the curve. *Hermite* patches (ibid, pp. 134) offer local control and continuity C1. Every patch is associated with a surface S and is defined by a matrix formed by the contour conditions; position of the vertexes, tangent vectors on these vertexes (first derivatives of the curve) and twist vectors (second derivatives).

$$\begin{pmatrix} S(0,0) & S(0,0) & \partial/\partial v\, S(0,0) & \partial/\partial v\, S(0,1) \\ S(1,0) & S(1,1) & \partial/\partial v\, S(1,0) & \partial/\partial v\, S(1,1) \\ \partial/\partial u\, S(0,0) & \partial/\partial u\, S(0,1) & \partial^2/\partial u\partial v\, S(0,0) & \partial^2/\partial u\partial v\, S(0,1) \\ \partial/\partial u\, S(1,0) & \partial/\partial u\, S(1,1) & \partial^2/\partial u\partial v\, S(1,0) & \partial^2/\partial u\partial v\, S(1,1) \end{pmatrix}$$

Fig 32: *Hermite* patch matrix.

In fig. 33 a *Hermite* patch is shown, with the most important information conveyed graphically (the curves displayed are the boundaries of the patch surface). The warping mesh is defined as a set of *Hermite* patches sharing the vertices (and tangent and *twist* vectors) with those adjacent. Those vertices are, in fact, the control points and by moving them the surface is redefined. The way patches are connected is illustrated by the example in fig. 34. Nine control points define a grid of 4 patches $P1, P2, P3, P4$. Analyzing this structure it ~~is~~ can immediately be seen that it is unable to offer enough control. For example, displacing the control point $v2$ results in the change of the contour properties of all the patches, because the tangential vectors of that point change and they all share that vertex.

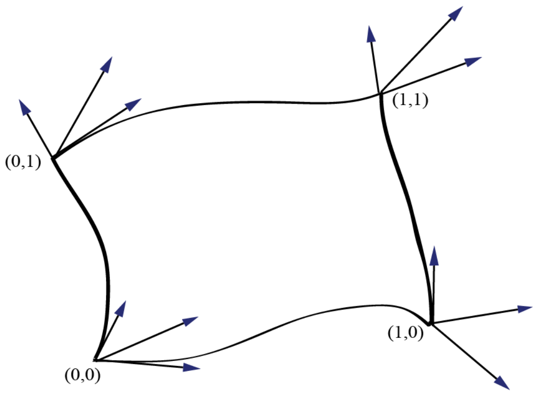


Fig. 33: *Hermite* patch

However, displacing the point *v1* would not have the same effect since it would only affect *P3*. The reason this happens is because the structure of the grid defines the curves of the surface from left to right and from bottom to top. For this reason modifying a point will affect the patch that contains it and also all the patches to the left and below, but not the rest of them. In other words, the local control is not symmetric to the point modified, which is a property that is not desirable.

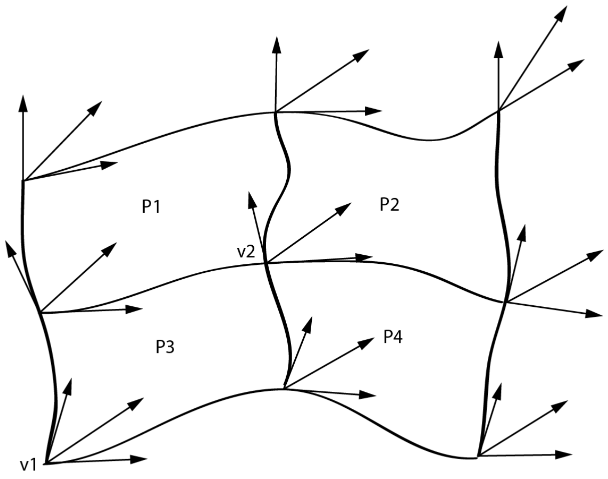


Fig 34: Grid of *Hermite* patches linked by their vertexes (control points of the grid).

A possible variation that was also implemented consists in defining four regions on the surface in a way that the point that is being modified sets the boundaries among them. This also has the advantage of avoiding unconnected patches on the borders of the surface. However, the price to pay is that the dependency will not be only on the positions, tangents and twist vectors of the vertices but also the last vertex that was modified. In fig. 35 there is a surface composed by 16 *Hermite* patches (25 control points) with the central point highlighted. If that point was the last one to be changed, the areas are determined in the way that is illustrated by the different colours (left-bottom, right-bottom, left-top and right-top). However, if another point is modified afterwards, these areas change and the shape of the patches change slightly. Even though the result is correct, this property is not desirable because local control is desired at all times. For that reason, the model chosen to define the mesh was replaced by another one; *Catmull-Rom* patches (ibid, pp. 165). Even though this is a similar model, it is more flexible and thus more appropriate for the task since the directional definition does not exist.

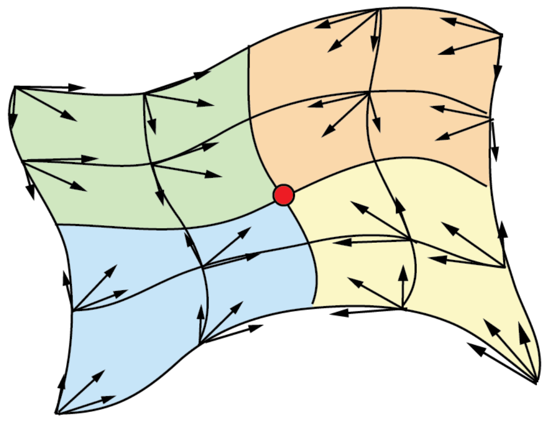


Fig 35: Grid of *Hermite* patches with definition depending on the last point modified.

Catmull-Rom patches are a straight-forward extension to a surface of *Catmull-Rom* curves. In analogy to the *Catmull-Rom* curve segment (which involves four points but only passes through the two interior points) a single *Catmull-Rom* surface patch is specified by 16 points. The patch is anchored at the four middle points and spans the area delimited by them (see fig. 36).

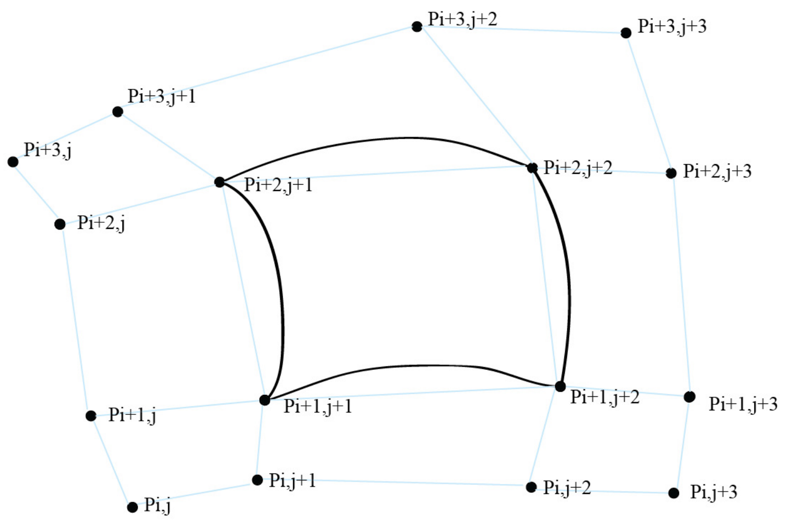


Fig 36: *Catmull-Rom* patch fully defined by 16 points.

Those 16 points define the contour conditions that can be expressed in a matrix (see fig. 37) that can be used together with a parabolic blending matrix (ibid, pp. 162) to determine any point of the surface. It is possible to take advantage of another property of the blending matrix, by setting the desired tension of the curves. Doing so, patches can vary from linear to highly curved just by adjusting that parameter.

$$\begin{pmatrix} P_{i+3,j} & P_{i+3,j+1} & P_{i+3,j+2} & P_{i+3,j+3} \\ P_{i+2,j} & P_{i+2,j+1} & P_{i+2,j+2} & P_{i+2,j+3} \\ P_{i+1,j} & P_{i+1,j+1} & P_{i+1,j+2} & P_{i+1,j+3} \\ P_{i,j} & P_{i,j+1} & P_{i,j+2} & P_{i,j+3} \end{pmatrix}$$

Fig 37: *Catmull-Rom* patch matrix (P).

Tension is defined as T = 1 - 2s; s being the parameter in the blending matrix that controls the lengths of the tangent vectors of the curve in the points (see fig. 38).

$$\begin{pmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Fig 38: Parabolic blending matrix (B).

The equation that solves the points of the patch can be expressed in a matrix arrangement (ibid, pp. 163):

$$P(u,w) = (u^3, u^2, u, 1)\, \boldsymbol{BPB}^T \begin{pmatrix} w^3 \\ w^2 \\ w \\ 1 \end{pmatrix}$$

It is very simple to connect as many patches as necessary into the same mesh. Adjacent patches will share the points as shown in the fig. 39a. The grid that defines the warping mesh is formed by 3 by 3 control points (4 patches). Patch *P1* is defined by the points coloured in red and yellow. Patch *P2* is defined by green and yellow points. In this example it is easy to see how to connect patches to form meshes of an arbitrary number of control points. However, there is a last detail to consider; the surrounding points that define the outer patches but that do not contact the surface (in the images of the fig 39a, all the points but the yellow ones) are not directly accessible by the user during the warping process (or, at least, they should not be, to keep the process as simple as possible). For this reason when one of the points on the boundaries of the mesh is changed, there should be an action associated to them as well. One possibility is to keep a constant distance (equal to the distance when the mesh has not been warped yet) between the points outside of the mesh and the points in the boundaries, which implies that there is an association between points and every time a point in a boundary is moved, at least one point outside of the mesh has to be moved as well (the association is shown in the fig. 39b). Experimentally, it was proven that this option is better when there are going to be overlapped regions between two projected surfaces projected by two different projectors.

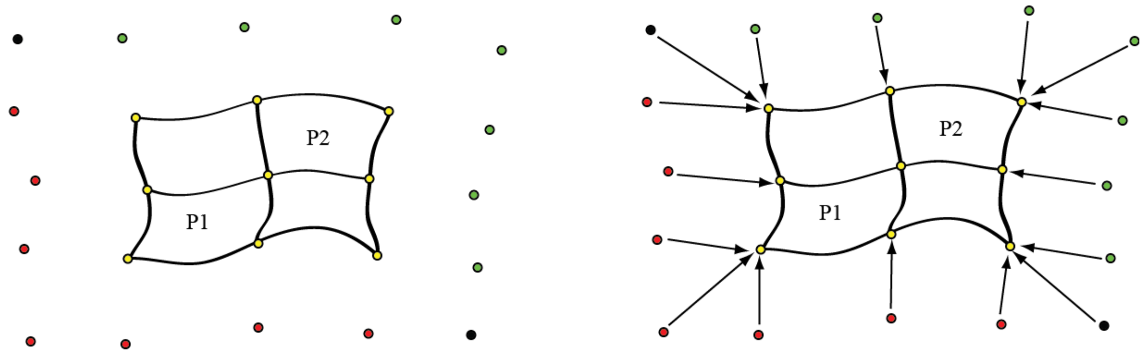


Fig 39: ***Catmull-Rom*** **patches connected on the left (a), and association between boundary and points out of the mesh, on the right (b)**

The explanation is that if the points outside of mesh should be the same as the ones projected by the other projector, and keeping a constant distance it attempts to approximate the behaviour of the mesh and the boundary as if it were continuous with the adjacent one. It would be possible to allow the user to configure those points individually as well, but the results would not be that different while the warping process would be more delicate and difficult.

However, another behaviour for the mesh on the boundaries has been implemented. Experimentally it was obvious that keeping a constant distance to the boundary points produced a 'spiky' behaviour of the mesh in the corners. While this can be useful at times, it can be a problem at others. For that reason, the proposed calibration program offers the possibility of collapsing the points outside of the mesh onto the boundary points, (see fig. 39b as a reference of corresponding points and how this occurs). The program allows a switch between the two modes ‘on the fly’. The shape that the mesh adopts doing so is significantly better for some calibrations that do not require overlapping regions, getting a softer contour.

Many other options that can make the warping step easier have been implemented as well. For example, the line strip that defines a row (or column) of control points should be as similar as possible to the resultant strip of adding or removing a point. The definition (in terms of polygons) of the mesh can also be determined by the user. In the Design chapter, some of these features, as well as the way the basic functionality of the mesh warper was implemented will be presented.

Another consideration to be made is the image that is to be displayed to perform the warping. The image has to show a simple pattern easily to align with the geometry of the screen. The most reasonable approach consists on a grid made of squares (as in fig. 18a). The idea is to align them and make them coincide with markers located around the screen. This aids the calibration process, and by changing the number of quads (the black and white squares) displayed on the warping mesh, the calibration can be achieved easily. The program proposed offers the possibility of adapting the number of squares. More information about the warping process can be found in the user´s manual in appendix 1.

3.2 Blending

The necessity of many applications for high resolution displays means that more than one projector needs to be utilised. In most cases a single projector of any technology is not sufficient for immersive virtual reality and scientific visualisation. In addition, their optics are such that it is not possible to align multiple projections accurately, so to avoid gaps, overlapping regions become necessary. The problem of this technique is that brightness is additive, and consequently in the area where the two projectors are projecting, the brightness is much higher because it is being contributed to by more than one light flux. Blending is the process of fading the images to black in the overlapping areas, so an imperceptible transition is achieved.

Without losing generality, the process will be explained for the simplest case, which is one vertical overlap between two flat images that share a region. The process is the same for situations where more than two images share an overlap or horizontal overlaps. In fig. 40 there is an example of two images of widths A and B that share a region of width C. It can immediately be seen that the resulting image will have width A+ B - C.

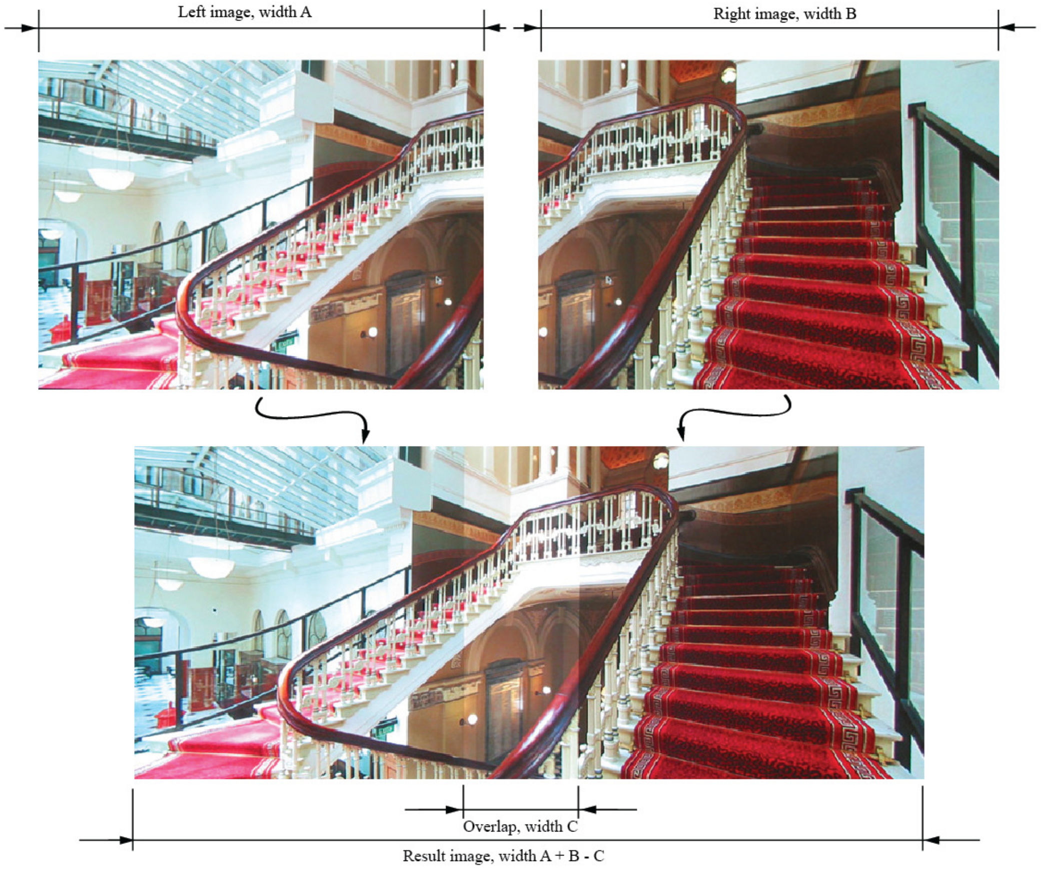


Fig 40: Left and right image and blended image with brighter overlapping region (no blending). (Bourke, 2004 [37])

To blend the images along the width of the blending area, it is necessary to multiply each pixel of the image on the left by an amount such as when it is added to its corresponding pixel of the other image (which is also multiplied by other amount). The result of this operation is the intended value of the pixel and is achieved by any function whose values lie between 0 and 1. Such a transfer function can be defined by basically any function, as the one proposed in [37] (Bourke, 2004) expressed just below.

$$f(x) = \begin{cases} a\,(2\,x)^{p} & 0 \leq x \leq 0.5 \\ 1 - (1 - a)\,(2\,(1 - x))^{p} & 0.5 < x \leq 1.0 \end{cases}$$

The function shows that for every pixel in the overlapping area, the factor applied for left and right images will be *f(x)* and *1-f(x)* (or vice versa). The parameters define an exponential function that can be tuned from being linear to highly curved. The factor '*p*' is the exponent of the function, while '*a*' defines the centre of the blend region (allowing more control in the middle of the blending area).If it is greater than 0.5, it will brighten, and if it is less than 0.5 it will become darker (see fig. 41).

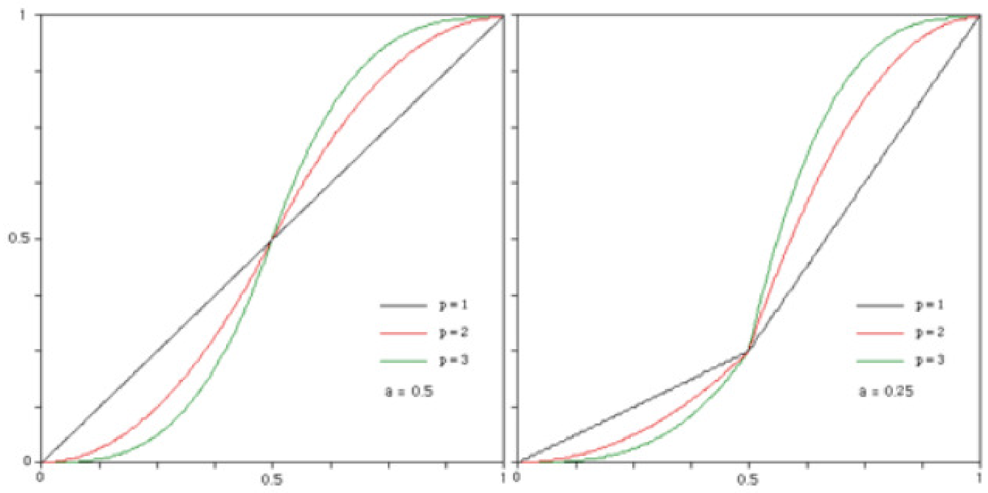


Fig 41: Blending functions, normalized coordinates. a=0.5 on the left (a) and a=0.25 on the right (b).

In the examples shown, at coordinate 0 (beginning of blending region) the pixels are multiplied by 0, so they do not contribute at all to the resulting blended image (the same pixel for the opposite image should be multiplied by one). In the middle of the overlap (coordinate 0.5), applying the functions in fig.41a the pixel´s brightness is reduced to half, so the other image should have this process repeated to obtain a level of brightness equal to 1. In fig. 41b the pixels are reduced to 25% of their brightness, which means they have to be reduced to 75% on the complementary image. Using this function, a blending mask can be easily calculated.

However, even though this is the basis of the concept, it is not enough to completely resolve the inconstancies in the image as a darker shade would develop in the blending region. The reason for this is that the described technique above, is adding pixel values. Each pixel is an RGB triplet and these are the values that are corrected with the exponential function. The problem is that what actually needs to be done is to correct brightness levels. To be able to do this, first a transformation that takes RGB pixel values to a linear space needs to be performed. This process is knows as gamma correction [38], and controls how pixels are mapped to brightness (typical values of gamma are from 1.8 to 2.2 depending on the projector).

Fortunately, it is enough to apply another exponential function to the blending function previously defined, becoming:

$$f(x) = \begin{cases} (a\,(2x)^{p})^{1/G} & 0 \leq x \leq 0.5 \\ (1 - (1 - a)\,(2\,(1 - x))^{p})^{1/G} & 0.5 < x \leq 1.0 \end{cases}$$

The process described graphically on the image 42 can be summarized as follows. First of all, the blending masks are calculated (using the exponential blending functions with the gamma correction). Secondly, to be able to apply the blending mask to the source image, the image's input colours have to be taken to a linear space so, again, the gamma correction is used. With these two images in a linear space, they can be combined without a problem. Once this is done, the inverse of the gamma correction has to be applied to take the colours back to the normal RGB space. When the two images are projected onto the same surface, the overlapping region should have the same brightness level as the rest of the projected area.

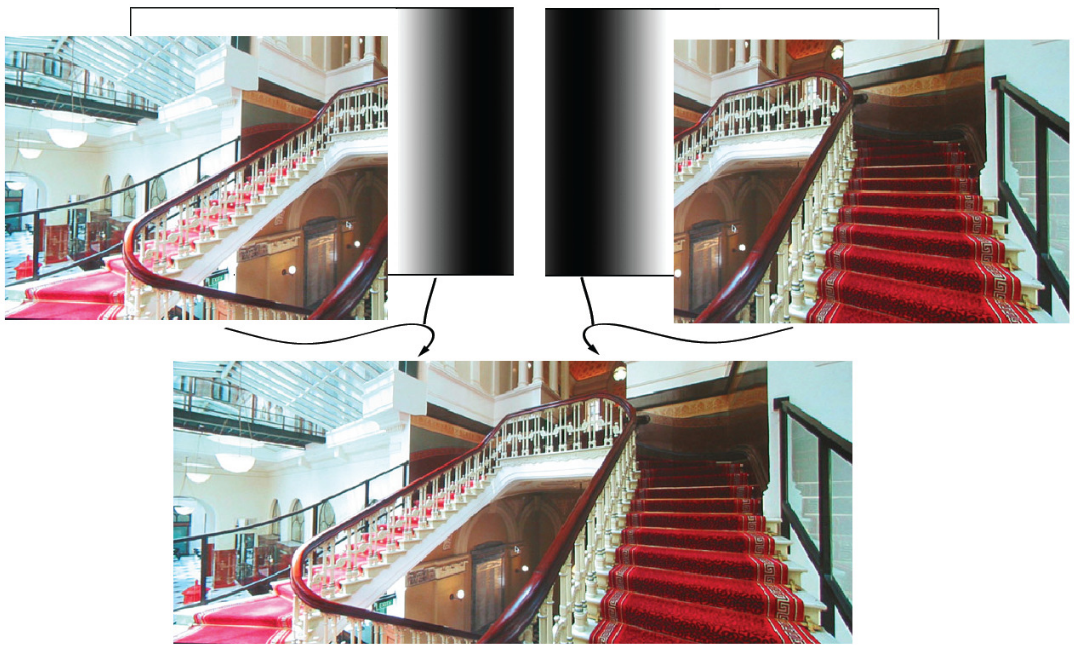


Fig 42: Blending with gamma correction applied.

The problem is that in real situations, there are many other factors that complicate the blending process. For example, projectors do not project with a homogeneous brightness. This becomes particularly evident when the projector is at an angle relative to the screen, is distant, or old. In fact, blending is really complicated when the projectors utilised have very different characteristics. In an ideal situation, projectors are able to project a homogeneous flux of light with the same brightness under any circumstance. In this case, the model that has been exposed is sufficient; otherwise the blending boundary (line that defined the limit where the image will not be treated) has to offer more flexibility in its configuration. In other words, the results achieved using a single vertical line, (or horizontal in case the blending region is at the top or bottom of the image), usually is not very satisfactory. For example, in a system like *AVIE*, the projectors are located above the screen and consequently the upper part of the screen appears brighter that the lower part. For this reason the blending regions should have a trapezoidal shape rather than rectangular. The image below (fig. 43) shows an example problem, where brightness in the resulting image in the overlapping region becomes higher than one, (in normalized levels of brightness that means that the image is brighter in the transition than the maximum that a single projector can offer). This is due to the fact that the maximum brightness of the projectors can be irregular in the borders of the image and after applying a blending function, that in theory would work, does not in real life.

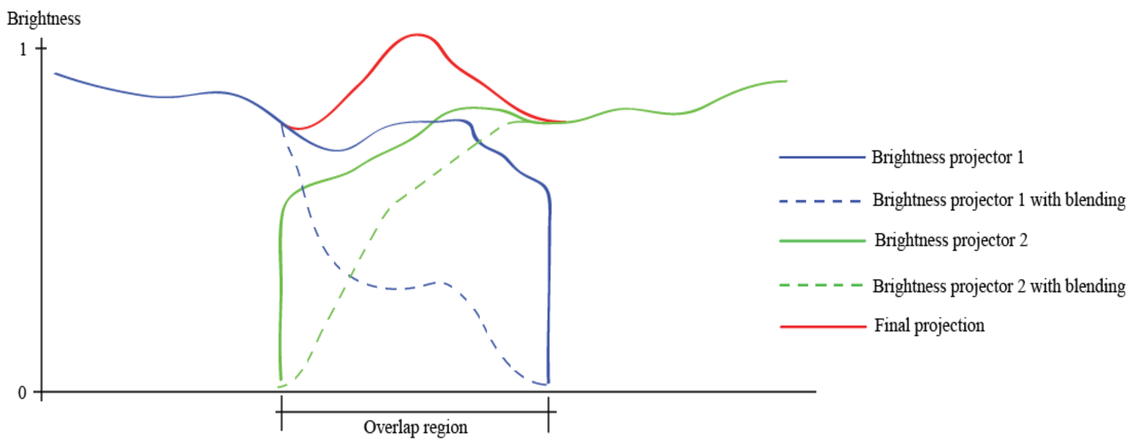


Fig. 43: Example of typical problem in blending region.

In order to aid in solving this kind of problem, the blending system that has been implemented offers the possibility of defining curved blending boundaries determined by four control points. This makes the overall calibration process much easier. Apart from that, for calibrating a system it is sometimes not enough to do it with white images. The explanation for this is that the exponential function for the gamma correction does not affect the maximum and minimum brightness levels. Thus an image with more chromatic information is needed. In the program a coloured pattern with significant colour transitions can be selected during the blending stage. It also offers the option of loading arbitrary images, chosen by the user. In the image below (fig.

44), two blending masks are applied to a white image (fig. 44a) and the colour pattern (fig. 44b). The meshes are not warped to illustrate clearly how the blending boundaries (displayed in red) can be curved. More information about the blending process can be found on the user´s manual in the appendix 1.

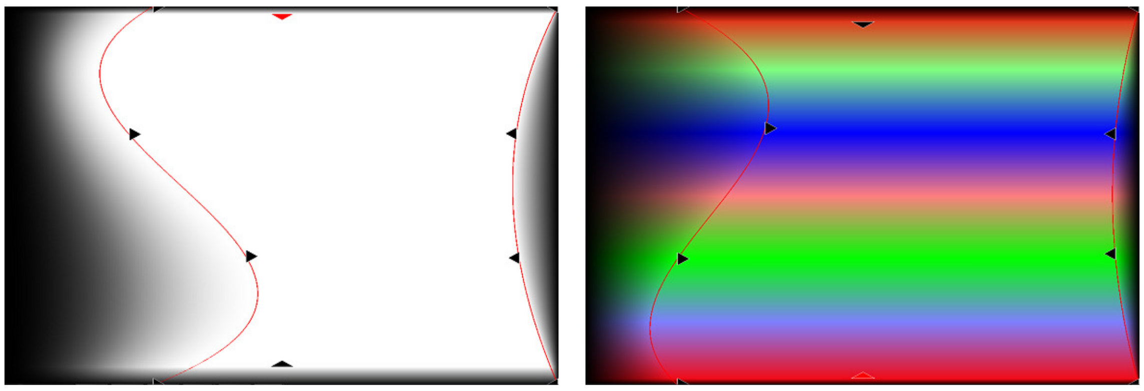


Fig. 44: Screen captures of two blending configurations, with the proposed calibration program.

3.3 Wide angle perspectives

A projection is a linear transformation that is defined in an n-dimensional Cartesian space that maps elements belonging to it onto a lower dimensional space, typically n-1. Mathematically, a perspective projection is a type of 3D projection that takes elements from the three-dimensional world and translates them to a two-dimensional world (plane). It makes sense then, that humans (being in a three-dimensional world) represent reality in flat planes as the most straight-forward and natural way. However, this kind of linear projections is often not appropriate and sometimes it is more interesting to use another kind of coordinate system, rather than the Cartesian one (i.e. cylindrical, spherical and so forth) as this chapter is going to justify why. This is exactly what happens in the case of wide angle perspectives.

3.3.1 Perspective projection problems

Linear perspective is a mathematical method for creating the illusion of space and distance on a flat surface. This kind of perspective (from Latin *perspicere*, ‘*to see through*’) was adopted as the most common model to represent reality as we perceive it though our eyes. A wide range of theoretical basis has been developed in arts, optics, and more recently, synthetic imaging. The basic idea behind linear perspective consists in representing the light that passes from a scene through an imaginary rectangle (i.e. the painting) to the viewer´s eye (see fig. 45).

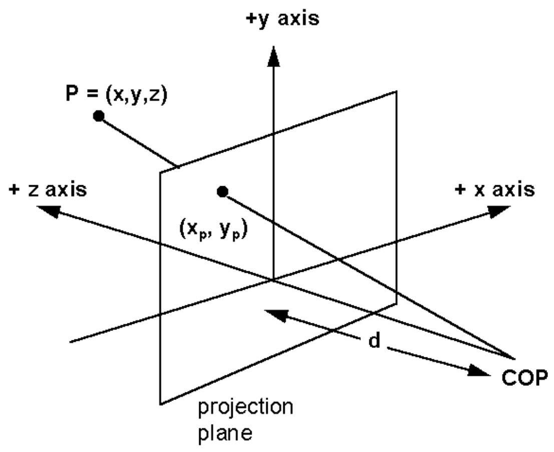


Fig 45: Simplest model of linear perspective projection (Lingard, 1995 [39]).

The first time a clearly modern basis of perspective was given was in 1021 by *Alhazen* (965-1039), an Iraqi physicist and mathematician that explained that light projects conically into the eye in his *´Book of Optics´*. However, *Alhazen* was only concerned with optics, and his theories were not used in drawing due to the fact that conical translations are mathematically difficult and very time consuming. By the 14th century, *Alhazen´*s book was available in an Italian translation and it is believed that during the early *Renaissance* it had an important impact in Italian painting. *Filippo Brunelleschi* (1337-1446) demonstrated in 1413 the geometrical method of perspective, used today by artists, which has been extended to many other fields, as for example computer graphics.

It is reasonable that linear perspective projections became a standard since they offer more realism than other approaches, such as orthographic projections (parallel projection where all the projection lines are orthogonal to the projection plane). They are very easy to implement with a computer, simply by defining a perspective projective matrix (Foley, 1997 [40]). Their use in computer graphics is so common that perspective transformations within the graphic pipeline has been extremely optimised (in fact graphic libraries such as *OpenGL* or *DirectX* only offer the possibility of defining orthographic and perspective projections). Perspective projection maintains straight lines as straight lines; they stay parallel if perpendicular to the viewer while converge to a vanishing point if they are not. While these properties seem reasonable, there are many limitations that cause this model not to be the most appropriate in all circumstances. One of the main drawbacks is the problem of foreshortening, something that

Leonardo DaVinci (1452-1519) had already realised. Perspective images are mathematically correct, but they appear distorted to us; objects near the edges of the image become stretched and/or sheared (especially in the corners). Relative depths between near and far objects also appear exaggerated compared to our normal viewing experience. The reason for this is that under normal viewing conditions (for example in a printed photograph) the field of view subtended by the image is far less than that used in the original photographic process. In fig. 46 extracted from [27] it can be seen how a ray from the viewer to the image represents a very different directional ray in the original camera.

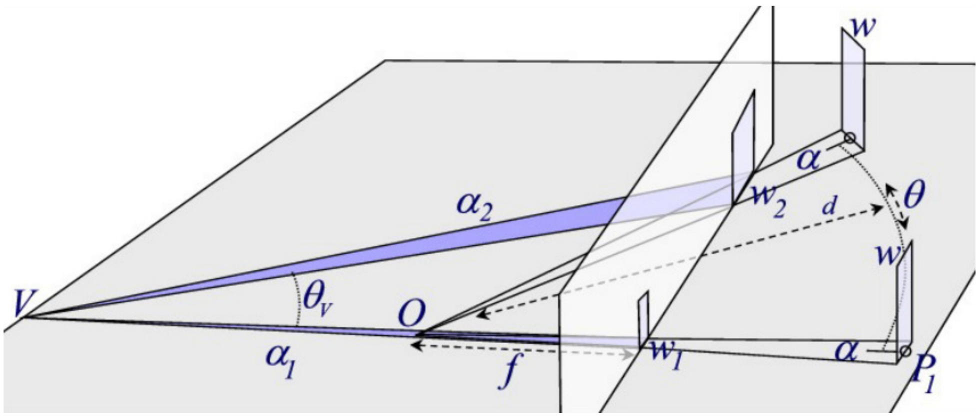


Fig 46: Graphic explanation for the foreshortening effect. (Liu, 2002 [27])

A narrow angle from the camera towards the edge of the image represents a much larger angle for the viewer. The opposite is true near the centre of the image; the same angle from the camera is much smaller from the viewer´s perspective. It is easy to see, using the example of a massive plane image that has been rendered with a wide angle planar approach with the observer located very close to that large plane. Objects at the extremities would be, in fact, extremely large when compared to the central ones. The reality is that they are also further apart from the user than those in the centre, so they would not appear so stretched or distorted when seen from the distance.

This idea has something to do with the concept of *anamorphosis* (from the greek '*formed again'*). Anamorphic images [41] require a special observation point. If the point of view changes (as in fig. 46) the images will seem distorted, but in fact the mathematical model employed is correct (see fig. 47). This effect causes non-uniform scaling and shearing distortions. At the centre of the image, it causes exaggerated foreshortening of distant objects.

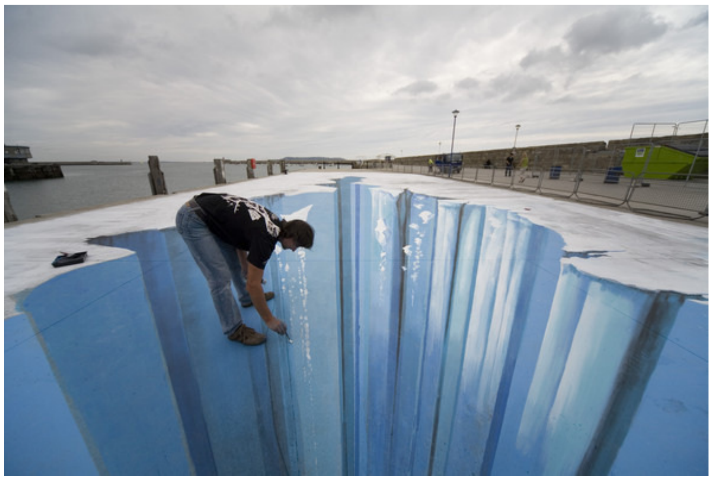


Fig 47: Street anamorphic art. From the artist point of view the image looks extremely distorted, while from much further apart the projection is correct (Edgar Mueller, 2008).

The deformation becomes even more evident in panoramic images with very wide angles. Objects appear stretched on the left side of the image, then compressed in the centre and finally, stretched again on the right edge. There is an example image that illustrates that problem below (fig. 48). The three objects are identical, and even though the two located on the edges are further apart, they look much larger due to the overstretching problem described above. For this reason, linear perspectives with wider angles than 60º become impractical.

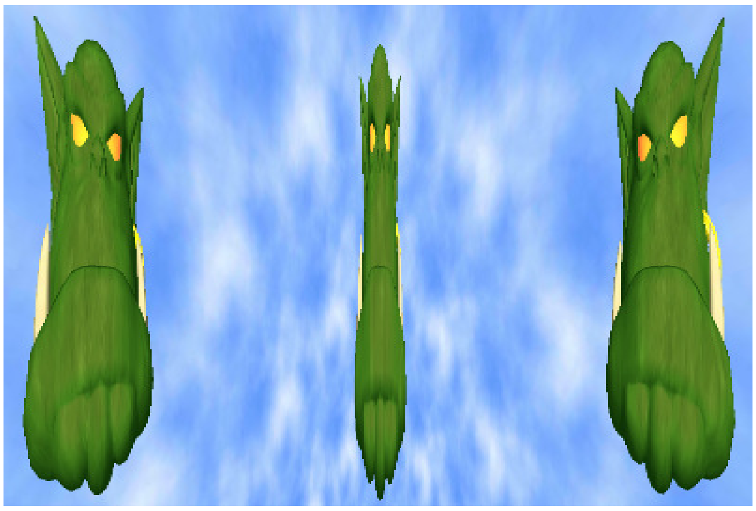
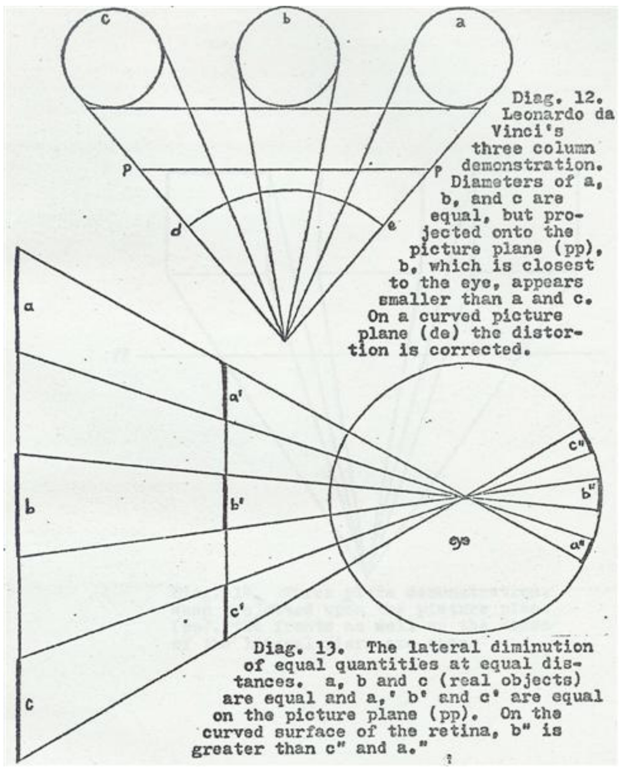


Fig. 48: 'Distortions' introduced by wide angle linear perspective (aprox. 170 degrees wide).

DaVinci illustrated the overstretching problem inherent to linear perspectives by using an example. He observed that proportionality is not respected between the distance of an object and its apparent size. He used the example of the three columns to prove his idea. When three columns of equal diameter are projected onto a flat plane (as when we use a perspective planar projection) paradoxically the columns that are further apart appear larger in the projection. Proportionality is only respected when objects are located in the same line of sight (for example, in the fig. 49, a column *b2* (not shown in the image) located behind *b* would look smaller than b) but objects that are not in that line of sight to not. In the meantime, he observed that when the projection is made onto a curve, (whose centre is on the origin of projection), the proportionality is respected, which makes much more sense. In fig. 49, the projection of columns *a* and *c* are smaller than *b*.



Diag. 12. Leonardo da Vinci's three column demonstration. Diameters of a, b, and c are equal, but projected onto the picture plane (pp), b, which is closest to the eye, appears smaller than a and c. On a curved picture plane (de) the distortion is corrected.

Diag. 13. The lateral diminution of equal quantities at equal distances. a, b and c (real objects) are equal and a,' b' and c' are equal on the picture plane (pp). On the curved surface of the retina, b" is greater than c" and a."

Fig 49: ***Leonardo Da Vinci´s*** **columns problem. (Collier, 1975 [42])**

Distortions grow with the apex angle (field of vision angle), so one may think that using a cylindrical section to project the image would help. It actually depends on the usage, because if the final image is to be displayed on a flat surface (i.e. a screen) ‘unrolling’ the cylinder onto a flat image may have undesired effects; such as straight lines drawn as curves. However, this is not a defect of a curvilinear perspective projection, but a problem of the flattening process. For this reason, if the field of view is not very large, a perspective projection still performs a great task. In other circumstances (when panoramic projections are needed, or the images are to be displayed onto curved screens), a curvilinear perspective projection is strictly necessary.

A curvilinear approach also seems more reasonable if we take into account the way the human eye works. A pinhole camera is the simplest model of camera that exists; it does not even need a lens. It is composed of a dark box with a tiny aperture that allows light in. Light passes through the hole and is projected upside-down onto the inner wall of the box. The human eye works in a very similar way, the most significant difference being that the 'wall' that the image is projected onto is curved (see fig 49).

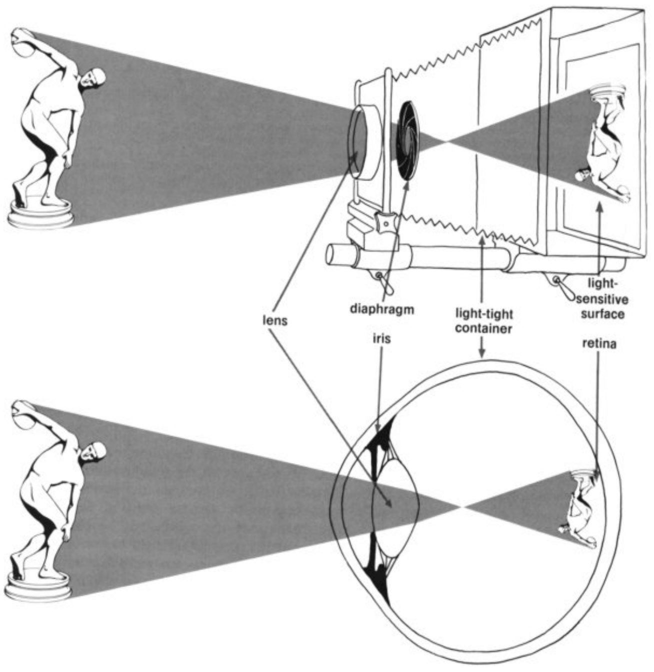


Fig 50: Comparative of elements between a simple camera and the human eye. (Estes, 1998 [43])

As stated in [44] (Glaeser, 1999), light projects in conic sections onto the human cornea. Even though human vision system has a horizontal field of view of nearly 180 degrees, human eye can only see sharply within a narrow cone with an apex angle of about 1 degree. When large fields of view are needed, eyes rotate quickly in order to produce several images of details of the scene, which the brain will combine into a single impression. This impression only corresponds to images produced by perspective projections onto the cornea of less than 30 degrees. What is important to highlight is the curved nature of the cornea, where all the photo-receptors are located. This characteristic should be taken into account when creating subjective perspectives, or in other words, when the scene is going to be seen by the user in first person (HMDs, fully immersive screen, and so on.).

However, it is important to keep in mind the purpose of the task and how the contents are to be displayed to the final user. It would not make sense, for example, to calculate projections in the most similar way to the eye that is possible if the images are not going to be presented in a fully immersive system (subjective perspective) but in a regular screen or similar.

If the goal is to calculate a subjective perspective, then it makes sense to be concerned by stereovision as well. According to what has been expressed previously (and obviating a multitude of details) eyes work just as a camera with a curved projection wall. If it is possible to find a projection type coherent with the eye system and it is presented to each eye within a layer covering the whole field of view, then this would be the first step for creating a truly immersive visualisation system. However, humans are also capable of perceiving depth by means of the disparity between the information obtained by each eye independently. Unfortunately, correct realistic perception is a process that involves many more factors that contribute in different ways, than just the action of presenting two different images to the eyes. In the next section, stereo for panoramic projections (in particular in curvilinear spaces) is analysed.

3.3.2 Multi-Directional Stereo

Stereopsis is the technical name for the retinal disparity due to the fact that humans, and many other animals, have two eyes. The popular belief is that this information is sufficient for the brain to infer depth information, but this has been proven false. Depth perception arises from a variety of depth cues, related with physiological and psychological factors. They can be classified in different groups [45], the first being monocular cues. These cues provide depth information when viewing a scene only with one eye.

- **Motion parallax:** It is the effect that happens when an observer in movement appreciates the relative motion of several stationery objects. To illustrate this, it is useful to think of the relative movement of objects when driving a car. Closer objects (i.e. light posts) move faster than objects further away (i.e. mountains). Some birds also take advantage of this property by bobbing their heads to achieve motion parallax.
- **Kinetic depth perception:** A dynamic change of the size of an object gives a hint of its movement and location in the space.
- **Relative size and familiar size:** Occurs when the size of objects is previously known, or when two objects located in different position have similar sizes.

- **Perspective:** Parallel lines converging at infinity aid with the reconstruction of the relative distance between two parts of an object, or landscape features.

- **Occlusion:** Objects covering each other provide information about relative distance.

- **Peripheral vision:** Parallel lines become curved at the outer extremes of the vision field.

- **Aerial perspective:** Due to light scattering by the atmosphere, objects that are a great distance away have lower luminance contrast, while objects in the foreground have higher contrast (usually modelled as fog in computer graphics). Also, the colour in distant objects is shifted towards the blue end of the spectrum. Some painters use warm colours (such as red, orange or yellow) to bring features forward.

- **Texture gradient:** Fine details can be appreciated only at short distances, while small features cannot be clearly differentiated at long distances.

- **Lighting/shading:** The way light falls in an object and the shadow it casts provide an effective cue for the brain to determine the shape and position of objects.

- **Accommodation:** This oculomotor cue provides depth information by the way the cililary muscles stretch the eye lens to change the focal lens (for example, the lens has to be thinner to focus on distant objects). The kinaesthetic sensation of contracting and relaxing the muscles is sent to the visual cortex, which uses it to interpret depth (only valid for distances lower than two meters).

In the same way, there are other cues that are supported by a simultaneous observation of a scene with both eyes.

- **Stereopsis:** Due to the fact that human eyes are located in different positions, the projections of the scene onto the retinas are different. By obtaining two captures of the same scene from different angles, it is possible to triangulate the distance to an object with a high degree of accuracy. Objects far away from the observer produce small disparities, while closer objects produce bigger disparities. In the Background chapter many methods to provide synthetic stereoscopic images were introduced.

- **Convergence:** In a similar way to accommodation, this is also an oculomotor mechanism to infer depth. When eyes focus on an object, they accommodate their direction to point directly to it. This ocular convergence is produced due to the fact that extra-ocular muscles are stretched with that purpose, and the magnitude of the stretch depends on the distance of the object focused. *Kinaesthetic* sensations are sent to the visual cortex that interprets the information. Convergence is effective for distances less than 10 meters.

Of these cues, only convergence, accommodation and familiar size provide absolute distance information, while the others are relative (they only make sense when compared to other objects). Even stereopsis is relative because a greater or lesser disparity could mean that the objects differ more or less substantially in relative depth or that the object is nearer or further away (the further away the scene is, the smaller the retinal disparity, indicating the same depth difference). Some other cues are inferred by our brain, for example when an object is being seen by the left eye, but not by the right eye. Our brain is able to integrate this information into the stereo construct with more or less accuracy. Individually, each cue gives indicative information useful for determining depth and some of them appear to have greater weight than others. However it remains unclear how they are weighted and integrated in the visual system by our brain. A summary of the most significant theories can be found in [46] (Boyd, 2000). Obviously, virtual reality applications are still far from being a perfectly believable experience, even though sometimes immersion does not need to be that extreme (depends on the purpose). The process of the human vision sight is very complex, and for that reason, immersive systems should take into account other factors apart from stereopsis. For example, working on enhancing motion parallax sensations, or a good colour and lighting setup can improve the experience in a large proportion.

There are two possible ways of achieving stereoscopic projections for linear planar perspectives. The first configuration of the camera pair is known as '*toe-in*' configuration. Its name comes from the fact that the two cameras are looking at a focal point (point of interest) and consequently they are focused inwards (see fig. 51a).

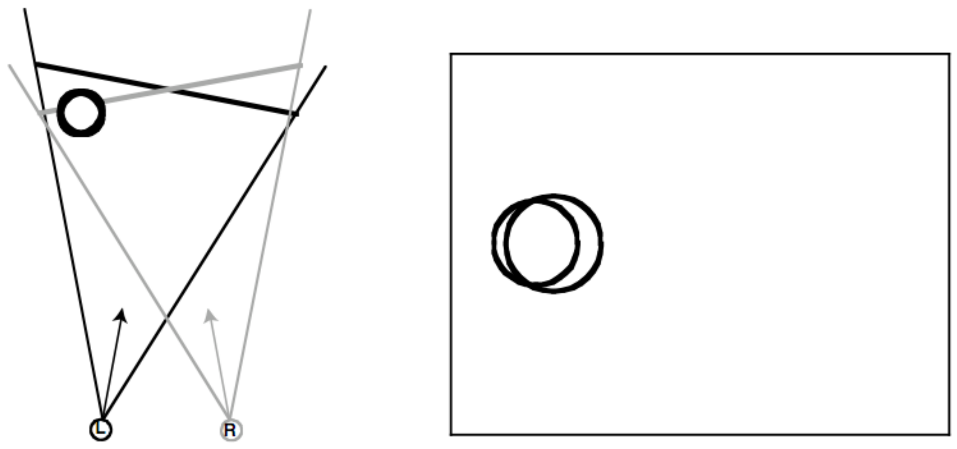


Fig 51: Problem of the 'toe-in' configuration for an stereo camera pair. Projection scheme on the left (a) and rendered image on the right (b). (Greenstone, 2004 [47]).

Even though the '*toe-in*' method mimics reality (since that is what eyes actually do when focusing in real world) this method has a serious problem. The fact that the projection planes are not parallel but crossed introduces vertical parallax distortions in the rendered image (see fig. 51b). This is due to the different distance of the origin of projections respect to their projection planes.

In the example shown, the sphere is located closer to the right eye´s projection plane than the left eye´s projection plane, so, the projections have different sizes. Even though, this difference in size seems to make sense (since the sphere is closer to one of the eyes), linear perspective projections exaggerate the effect on the edges. The 3D effect would be difficult on the eyes, especially in those regions. Normally, in planar perspectives, the best 3D effect is achieved when there is not parallax scaling but only horizontal shifting. In order to achieve this, off-axis projections are employed. By configuring the stereo pair with non-symmetric view volumes we can force the cameras to share on the projection plane, getting rid of the undesired scaling (see fig. 52).

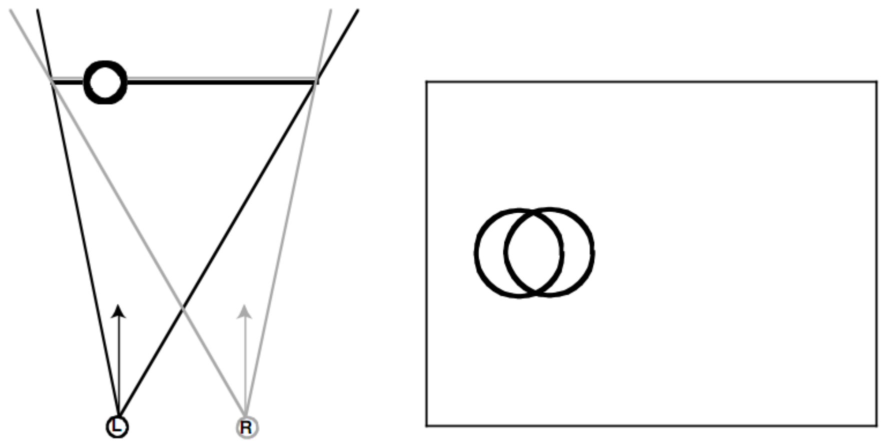


Fig 52: Off-axis stereo pair configuration on the left (a) and rendered image on the right (b). (Greenstone, 2004 [47])

Results produced by the off-axis configuration are based on a camera model that is not similar at all to our eyes, but it works fine for planar projections. First, in an off-set configuration, eyes are always focusing infinity, because sight lines do not converge as in the 'toe-in' configuration. On the other hand, the non symmetric definition of the view volume, force the eyes to share the projection plane which is not what happens in our vision system (we have two different retinas). However, the vision system and the combination of depth cues is a very complicated problem in natural vision, as it has been discussed in this chapter. Consequently, it is not possible to model how it works element by element, especially using a planar approach. Off-axis projections produce very satisfactory results comfortable to watch narrow fields of view.

However, some additional sacrifices have to be made, especially if several users have to share a display to view the information. A stereographic pair depends on the specific position of the eyes relative to the virtual world, so either the scene is calculated entirely for that user or another stereoscopic method has to be used. Also, in immersive systems, wide fields of view are often necessary. Justifications of why panoramic images are better displayed on curved screens have been given throughout this report, but summarising planar projections introduce distortions when wide angles are used. These are actually not distortions since, mathematically, the correct information regarding depth can be extracted. The problem is just that human vision system does not work in this way. Another aspect is that immersive systems should be able to ‘surround’ the user with a seamless projection of the artificial world.

Let´s assume for the moment that it is possible to calculate a stereo image for a single user utilising a dome, (see fig. 24) even though it has yet to be explained how to do this. Let us also assume that the head of the user is located exactly at the point for which the scene has been rendered or that a head-tracking system is being used. Unlike cameras, human eyes can move and re-focus, at it is needless to move the head. If the tracking is not being done at eye level (eye-tracking instead of head-tracking) an additional problem has to be faced. As eyes can move freely, the projection is calculated for the user´s head position is not strictly valid anymore they are not looking forward. Being realistic, in an immersive system with a large screen, users will always move their heads and also their eyes. Simplifying the above, not every system is multiuser and may or may not have a head-tracking system, so consequently the scene must be valid for many users, for example in an *AVIE* system (see fig. 27). For this reason, it is more convenient to use a new type of stereo projection, also called multi-directional stereo or omni-stereo projection, which is perfectly suitable for curved screens and panoramic scenes. The following sections analyse how to achieve this kind of projection using two different approaches.

3.3.3 Projection warping

Even though this approach has not been implemented for the system proposed, it is of particular interest to include it in this Analysis chapter, due to the fact that it supports the justification of the methods finally employed (multi-slice and curvilinear projection in shader).

The basic idea is to distort a planar perspective projection. In [48] (Bayarri, 1995) a method in two steps is proposed. The first step consists in projecting the vertexes of the objects onto a curved surface (cylinder or sphere), and from there onto a flat plane. While this method is suitable for rendering wide angle images since it avoids overstretching problems, it does not create real curved projections. It is easy to see why in the example in fig. 53. A projection hemisphere *PS* with radius *d* is defined with origin *O* in (*0, 0, 0*) as well as a projection plane *PP* of equation *Z=d*. The limits of the projection on the left and right are *l1* and *l2* respectively, that in this case are both equal *r*. For this example, the vertex to be projected *V*, is located on the plane *Y=0*, so the problem is reduced to 2 dimensions. The projection *Vs* onto the sphere of any point with coordinates (*a, 0, a*) will be $(r/\sqrt{2}, 0, r/\sqrt{2})$ that, in any case, is located at the point 0.75 in absolute coordinates of the sphere ring (it forms 135º out of 180º). However, the projection *Vp* onto the plane *PP* from *Vs* is $(r/\sqrt{2}, 0, d)$, that represents approximately 0.85 normalised to the distance between *l1* and *l2*. That difference (0.75 against 0.85) indicates that the correction that has been made is not translated into a real spherical projection.

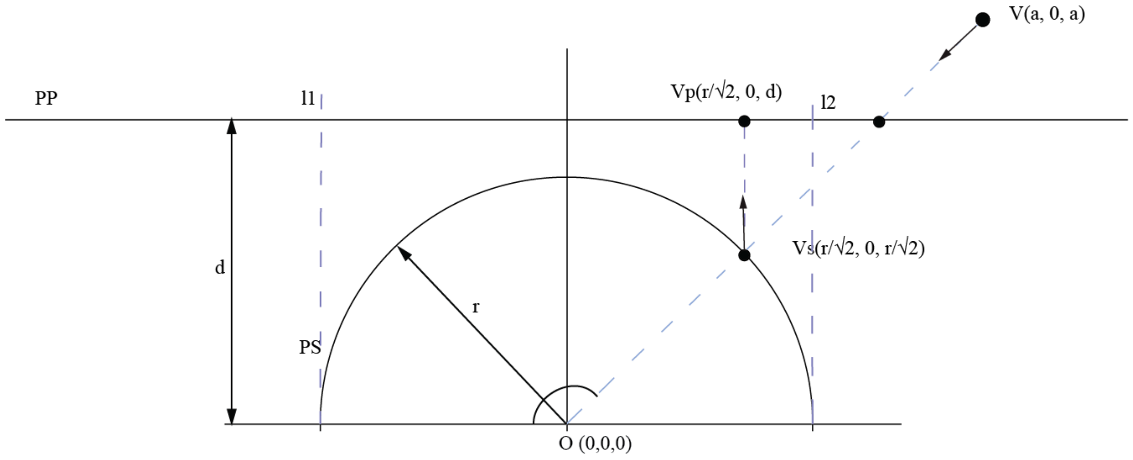


Fig 53: Curved projection in two steps. (Bayarri, 1995 [48]).

If the projection from *Vs* to the plane *PP* is not done orthogonally to the plane, but depends on the intersection point itself it is possible to control the distortion by modifying some parameters (distance *d* and the limits *l1* and *l2*). A very similar idea that extends the possibilities of the distortion is proposed in [49] (Yang, 2005). In this case, arbitrary surfaces defined as a polygonal mesh (projection lenses) can be used so the projections become distorted according to the contact point on the surface and the focal length (see fig. 54).

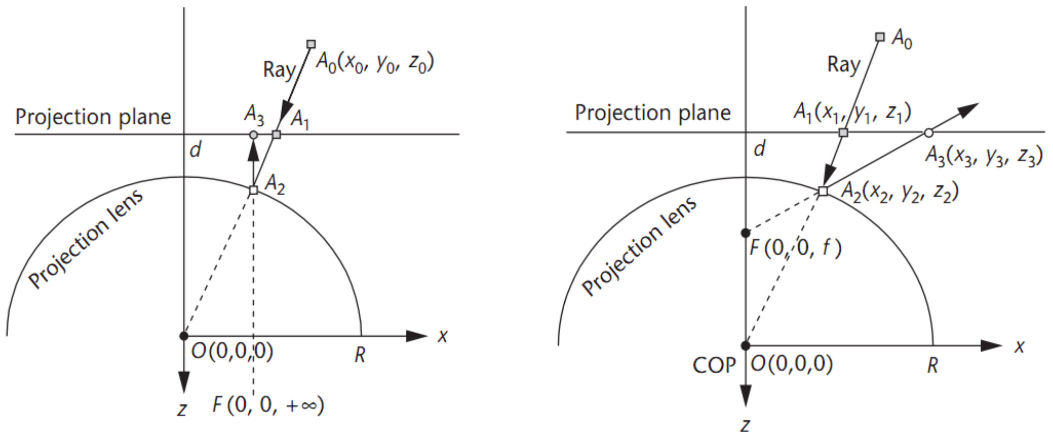


Fig 54: Curved projection in two steps. (Bayarri, 1995 [48]).

While these methods based on distortion lenses can produce very different effects, they are perhaps not the most straight forward way of achieving a 'correct' non-planar projection, since parameters such as the focal length of the lens dictate the process and they are difficult to set.

Other techniques take advantage of *cube-maps* [50]. A cube-map is a set of 6 square textures that fit together like the faces of a cube. Together, these six images form an omnidirectional image that it is used to encode environment maps. However, as each face is rendered independently as a perspective projection, there is not a smooth transition between the textures (see fig. 55). In the example, it is very obvious if the lower face (with perfect squares) is compared to the rest or the transition areas as highlighted (object´s shade).

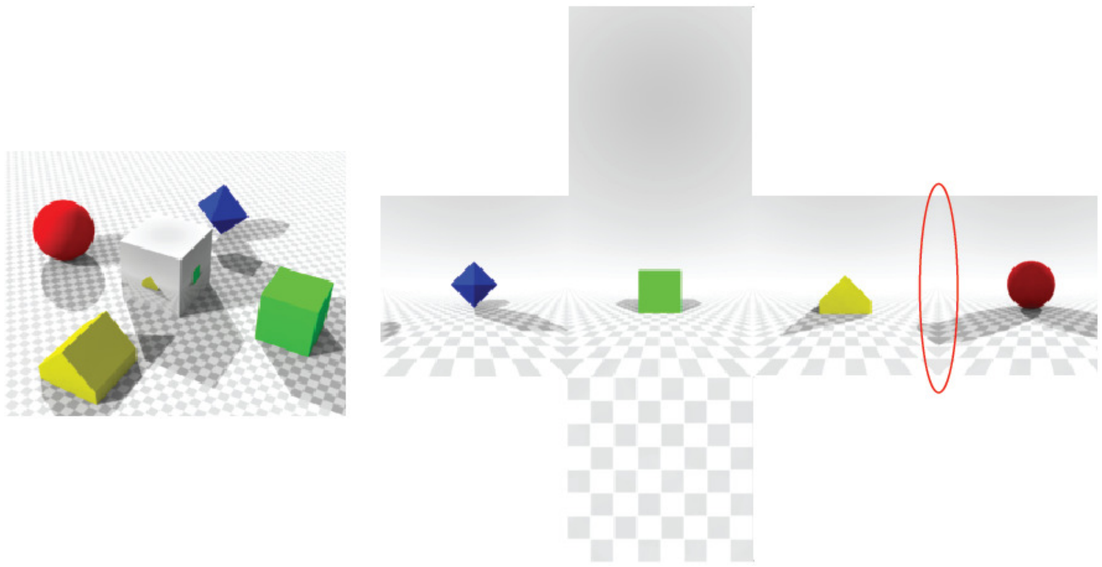


Fig 55: Texture images for a *cube-map* illustrating the lack of smoothness in the transition between the different faces (from *Wikipedia*)

However, the implementation of *cube-maps* is very simple and are highly optimised (*Nvidia* 2004) in graphic cards since they are used for many different purposes (i.e. global illumination) involving environmental information.

In [51] (Bourke, 2010) a cube-map (in fact, just 4 of the 6 faces) is adapted to be used in a hemispherical display like an *iDome* (see image. 24). The technique consists in tessellating a textured cube, so it can be deformed until it becomes hemispheric. Once that is done, the mesh is flattened according to the equiangular fisheye projection equations [52]. The final step consists in warping the fisheye projection to adapt it to the curved surface (see fig. 56).

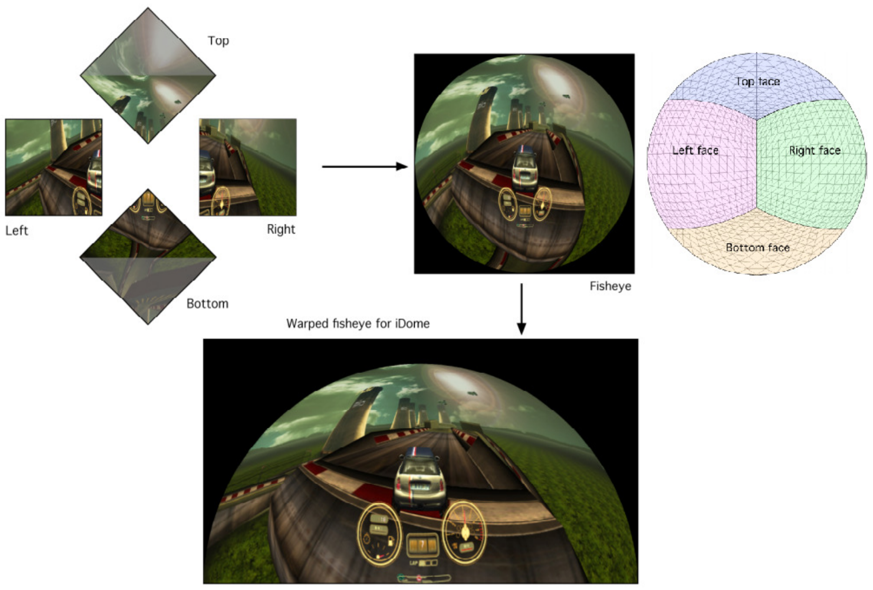


Fig 56: Adaptation of cube-map faces to curved displays. (Bourke, 2010 [51])

A similar idea is proposed in [53] (Trapp, 2008). The scene is rendered onto a cube-map as well, but this time cylindrical and spherical mapping functions are defined. For this reason, once the cube-map is available the process is reduced to create a texture within a fragment shader that defines the corresponding curvilinear projection. This projection can be derived by determining a cube-map sampling vector (in *cube-map* space), for each fragment of the texture. In other words, every pixel of the texture of the cylindrical projection is associated to one pixel of the cube map.

The problem of these methods is that they use a *cube-map* to derive a curved projection. Cube-maps are normally defined in terms of 6 planar projections, and for this reason, the result in not as accurate as it should be since a curvilinear projection has been extracted from a linear one (even though in most cases they are visually correct). For the same reason they do not support omnidirectional stereo, because even calculating two cubes, (for left and right eye), the linear projections are defined for the view directions perpendicular to the faces of the cube, and not for every direction. Another problem is that the process of deformation of the tessellated *cube-map* or the mapping functions may deteriorate the final quality of the image if the resolution is not high enough. Also, calculating cube-maps dynamically can be computationally expensive.

Finally, a method for creating piece-wise perspective projections was proposed in [54] (Lorenz, 2009). With this method, customable perspective deformations are supported (useful to create similar effects to [49]) but in this case the deformations are introduced in object space, by means of utilising geometry shaders. This method also supports cylindrical projections; however it is achieved by using multiple planar projections to approximate a cylindrical view volume. This technique can also be implemented in many ways, as it is the case in the system proposed. The next section will discuss this technique.

3.3.4 Multi-slice approach

One possibility that can be considered, is to obtain a curvilinear projection by means of approximating it with small flat projections (Simon, 2004 [55]) as shown in fig. 57. By doing this, if using enough flat projections, it is possible to avoid visible overstretching deformations and provide omnidirectional stereoscopy.

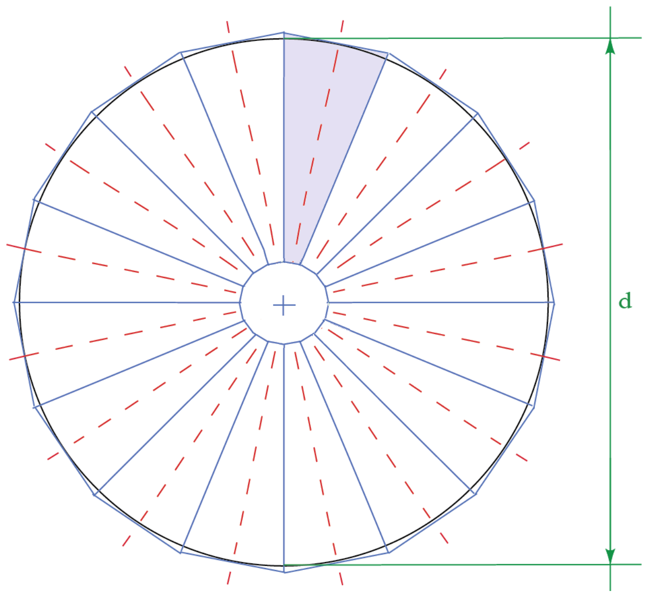


Fig 57: Scheme of a cylindrical view volume of diameter d with the Multi-slice approach.

In the image (fig. 57) a cylindrical view volume is approximated by 16 slices of 22.5º. Probably the major drawback of using multiple planar view volumes is that if not enough of them are utilised, discontinuities are obvious to the viewer. The problem is that, in general, the more (and consequently, the thinner) view frustums that are used, the more computational load is introduced to the system (see chapter Results).

A first slice renderer was developed in *OpenGL*, in order to experiment and compare the different results obtained depending on the number of slices. In the image below (see fig. 58) a simple scene is rendered for a 180º field of view using 100 slices (1.8º per slice), and the different slices are shown. By creating a slice-camera class, and using frame-buffer objects [56], the implementation of such a program is very straight forward. The basic idea consists in defining as many cameras as slices (or make rotate one to take the different captures) and associate every portion of the scene to a texture. Each one of these textures has to be composed into a single texture, forming the final image. In fig. 59 the process is illustrated. In the next chapter there are more implementations details of how to implement this.

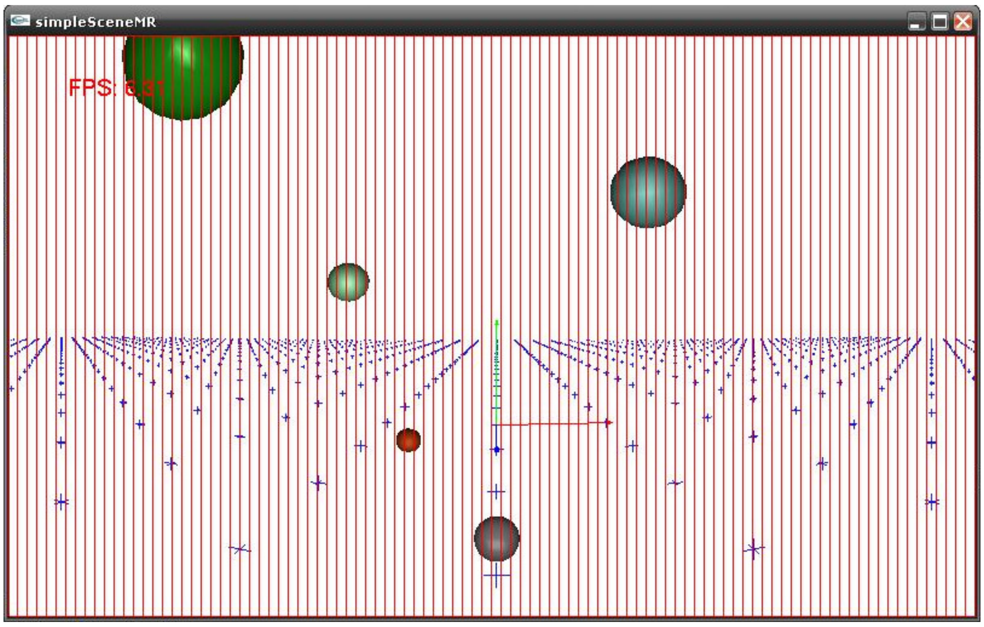


Fig 58: Screen capture of a slice-renderer prototype in *OpenGL*.

If the camera is not located in the centre of the screen, the rotation cannot be done in regular intervals around the vertical axis. In this case, rotations are not constant and off-axis projections are necessary to adapt the projection planes of the view volumes to the screen approximation, as show in the image (fig. 59).

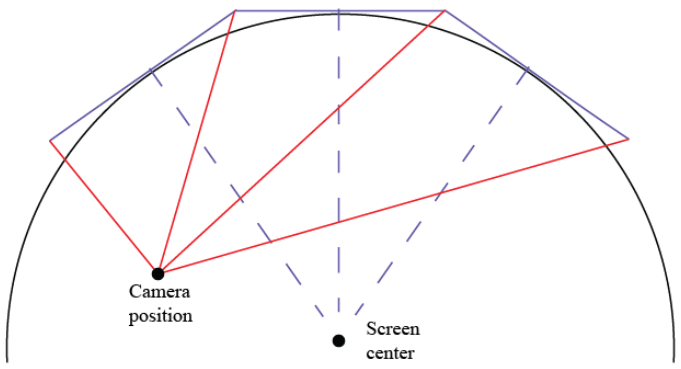


Fig 59: Off-axis projections for a camera not located in the centre of the screen.

Nonetheless, there are many details to consider for rendering an omnistereoscopic image. First of all, a configuration for the stereo camera pair has to be chosen between ‘toe-in’ (see fig. 51) and off-axis (see fig. 52). In [57] (Bourke, 2006) the two methods are discussed. In both, the basic idea consists in locating two cameras displaced horizontally, which are then spun around the vertical axis, taking captures of the scene in multiple rendering passes. Another detail to consider is that the interocular distance is more related to the scale objects rather than the actual human eye separation. For example, if a virtual model is on the scale of the solar system, to achieve depth effect, 6 centimetres would not be enough, but much larger distances should be used (i.e. on the scale of a planet´s radium).

Toe-in is interesting, because the set-up can be done in a way so that the parallax is controlled. The parallax is defined by the difference (separation in pixels or metres in screen space) between the objects projected onto the screen. A point belonging to a virtual object located on the surface of the screen (see fig. 60) should not show any parallax since the projected image of the point is physically in that location, and thus, no additional three-dimensional illusion is necessary.

In order to do this the view direction vectors of the cameras are no longer parallel to each other but rotated inwards. The intersection between the two view direction rays determines the distance zero parallax. As aforementioned, ‘toe-in’ cameras introduce vertical parallax towards the corners of the image due to the fact that the projection planes are not coincident but cross each other (see fig. 60). For this reason, this configuration is only feasible if the strips that form the final image are very narrow and no perceptible vertical parallax is introduced. The problem is that this is computationally very expensive, and discards this configuration for real-time applications. A few tests were performed in order to see which was the quality level achieved while maintaining good performance. No satisfactory results that could be employed for real time applications were found, and consequently, off-axis projections have been used.

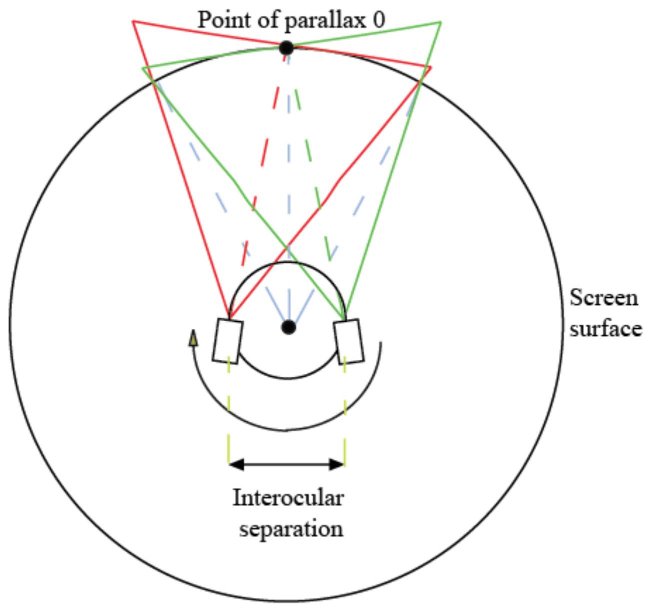


Fig 60: 'Toe-in' configuration, with parallax 0 point set on the screen.

Off-axis projections offer good stereographic results since with this configuration the projection planes are coincident. However, in order to do that parallel view directional rays are needed. This implies that the cameras are focused towards infinity, and consequently there is not a zero parallax point (see fig. 61).

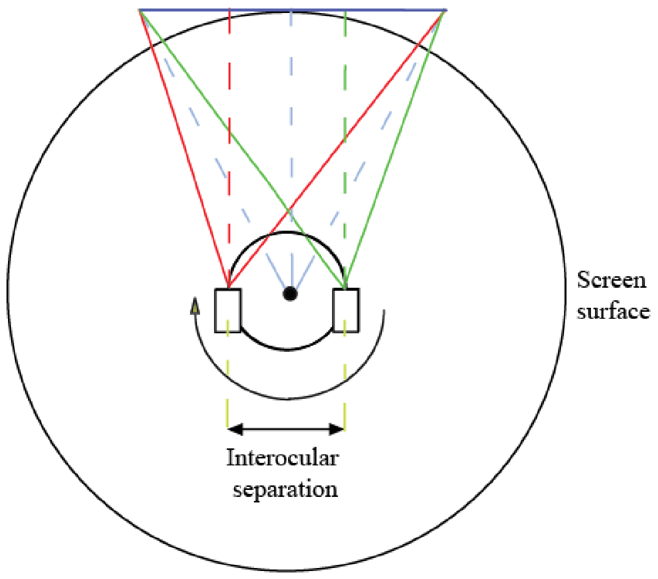


Fig 61: Off-axis configuration, with parallax 0 point set on the infinity.

In order to control the distance f_0 in the rendered scene in which objects will appear to be at the projection screen depth, post-processing of the final panoramic image is necessary. Objects closer than f_0 will appear to be in front of the screen (with negative parallax) while objects further than f_0 will appear to be behind the screen (positive parallax). Unlike perspective stereo pairs where horizontally shifting the images with respect to each other to control zero parallax introduces occlusion errors, stereoscopic panoramic pairs can be shifted without adding any additional error [57] (Bourke, 2006). The geometry for calculating the relative shift between the image pairs in order to achieve a particular f_0 is shown in fig. 62.

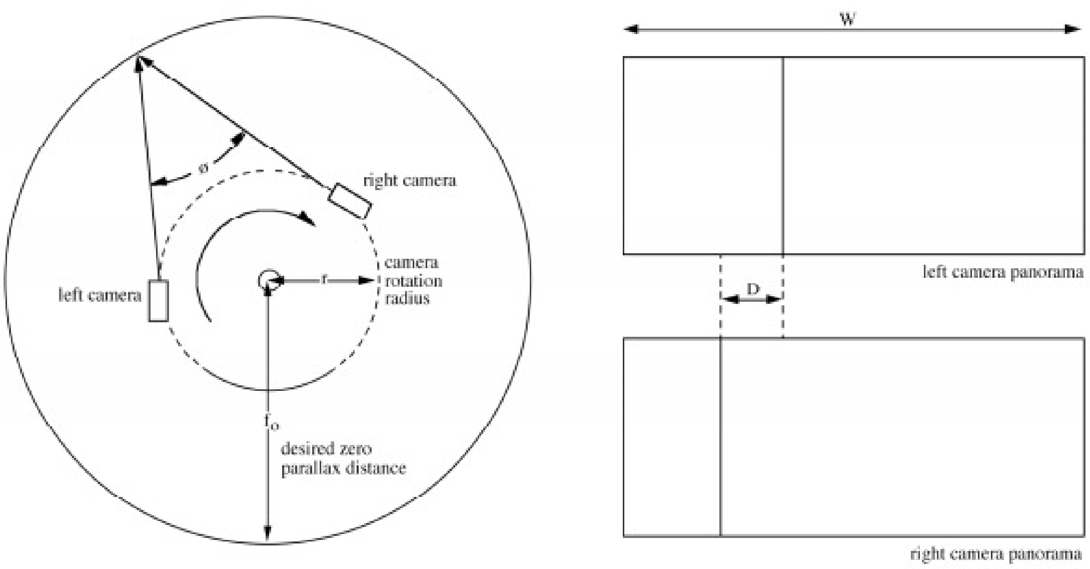


Fig. 62: Geometry for determining the distance to shift the panoramic pairs with respect to each other in order to achieve a particular zero parallax distance (Bourke, 2006 [57]).

The angle $ø$ between the left and right cameras such that an object is the desired zero parallax distance will be coincident on the left and right panoramic images is given by:

$$ø = 2\, asin(r / fo)$$

where r is half of the camera separation, and consequently the number of pixels D to shift the final panoramic images horizontally is:

$$D = W ø / (2\pi) = W asin(r / fo) / \pi$$

Where W is the width of the whole panoramic image measured in pixels.

As the calculations have been done for a full panoramic image, the shifting is performed in a circular fashion and those parts of the image that are shifted past the right hand edge of the panoramic image reappear on the left and vice-versa. The problem is that in some systems (like *AVIE*) the whole image is not available since it is rendered by sectors and consequently the shifting becomes a bit more complicated. A possibility is to render a wider field of view, in order to have access to the adjacent pixels locally.

The off-axis approach produces better results than the 'toe-in' when not many slices are utilised. However, the incorporation of multiple slices slows down the rendering process so there is a trade-off between performance and quality that has to be considered (see chapter Results). The major problem of the multi-slice approach comes from the fact that is an approximation of the real curved surface with facets. Discontinuities can be appreciated easily, especially when small objects are located very close to the camera (fig. 63).

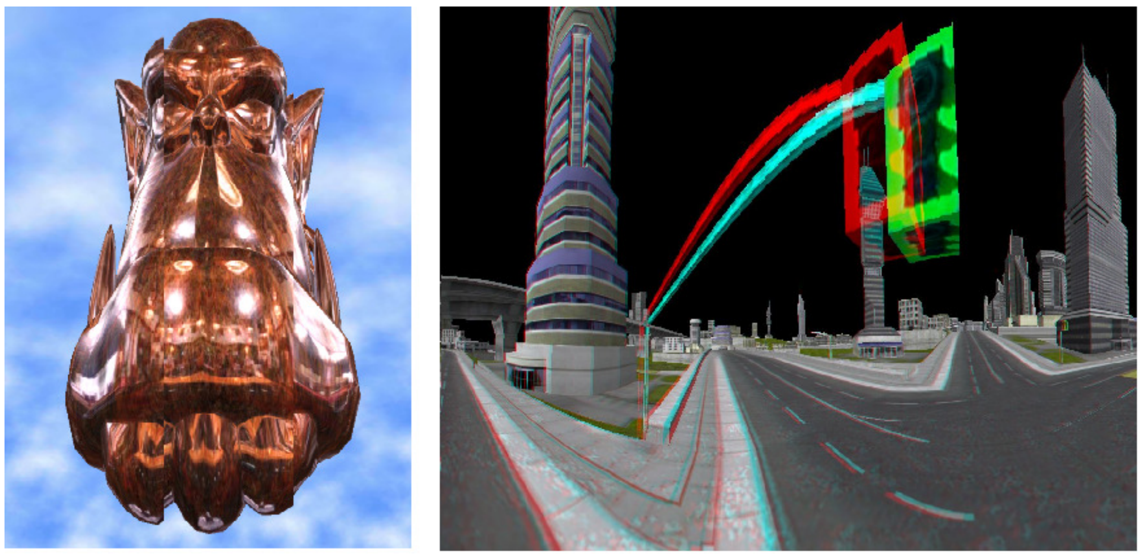


Fig 63: Artifacts produced by the slices. On the left the object is very close to the camera and small and on the left the discontinuities become obvious in the closer object (Lorenz, 2009 [1063]).

It is easy to see why the distortions happen with the example in fig. 64. The object represented within the view volume cannot be seen at all by the left eye in the left image (a), and it is fully contained in the left eye´s volume on the right image (b). The right´s eye view volumes are coloured in yellow. The green sphere is partially contained in the volume of the left image (a) and partially on the right (b). The problem is that the difference between the orientations of the view volume is very large, as is the difference between the position of the eye. For this reason the projection of the object differs significantly between them.

Even if there is not a geometrical discontinuity, sometimes the transition between areas is also evident. This can occur when objects with very homogeneous shape (i.e. sphere) are moving around the screen only horizontally. If the approximation is too coarse it is possible to see how the sphere becomes stretched out in the transition areas. Omnistereo quality is also affected since there are a finite number of defined view directions (as many as slices), being the transition areas where the disparity errors achieve their maximum. The artifacts become more obvious when some special effects need to be applied (i.e. volumetric fog, smoke or blur effect). The multi-slice concept can be generalized to spherical surfaces by varying the angle of the projections vertically as well, in fact if enough slices are used appropriately any general surface can be approximated, but this normally implies a computational overload that is too expensive. The lack of a smooth and seamless projection area (in terms of view frustums) is prone to cause problems. However, depending on the application, geometry displayed, and effects applied, multi-slice can still be a good option.

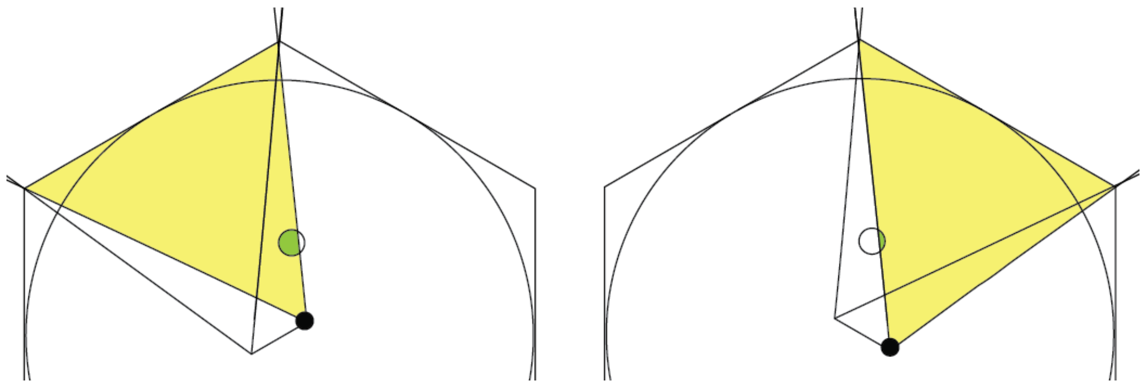


Fig 64: Very different view volumes can contain parts of the same object.

3.3.5 Shader-based approach

As it has been described in the previous section, the problem of approximating the surface with planar projections does not produce panoramic images as accurately as a real curvilinear projection. In addition, using too many slices affects the performance too much. However, they represent the most straightforward solution because the current graphic hardware does not support curvilinear projections natively yet. The graphics pipeline utilises projection matrices to perform the linear transformations between the different spaces (for the models, the view and the projection). In fact, as aforementioned, graphic libraries as *OpenGL* and *DirectX* only allow the programmer to specify the projection matrix as a linear transformation, either perspective or orthogonal. For this reason, if the transformation is to be done in a curvilinear fashion, the fixed functionality of the graphic pipeline has to be modified for the purpose.

Advantages of using a curvilinear transformation are obvious. Firstly, the results are accurate and it is needless to use multiple rendering passes in the pipeline (unlike the multi-slice). Secondly, as the projection is to be designed by the programmer, the distance of parallax 0 can be customised. Lastly, stereo (for a single user system) or omnistereo (for a multi-user system) will also be much more precise.

In order to modify the functionality of the pipeline in a way vertices are transformed in a curvilinear way, a vertex shader is used. The process can be seen as a substitution of the linear transformation matrix that is normally sent to the vertex shader by a different way of calculating the transformations. In order to perform a curvilinear transformation, a basic requirement is necessary; vertices cannot be in a linear space but are required to be in a curvilinear one. In case a cylindrical projection is to be performed, Cartesian coordinates of every vertex need to be converted to cylindrical coordinates (i.e. for *AVIE*). In the same way, spherical coordinates are necessary for a spherical projection (i.e. for *iDome*). Information regarding change of coordinate systems (between cylindrical/spherical and Cartesian) can be found in appendix 3.

The simplest idea to create stereoscopic curvilinear projections consists in using a camera pair with a horizontal off-set, rotated to achieve a '*toe-in*' configuration [58][59]. The angle is set in a way the camera view vectors point towards the distance of zero parallax (fig .65a), where the two sight lines find each other. The major problem is that the surfaces no longer match each other and consequently the correctness of the parallax information is completely lost in some situations (fig. 65b).

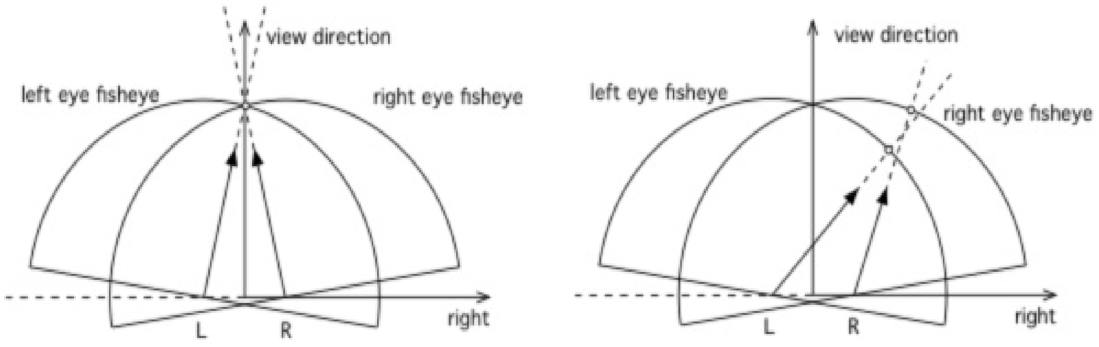


Fig 65: Coarse 't*oe-in*' curved projections (Bourke, 2009 [1082]).

Such a problem clearly shows that coincident projection surfaces are also necessary in the curved approach, as occurs with the multi-slice. Taking this into account, vertices should be projected onto the unique curved surface by intersecting with the line that passes from each eye to the object (normally the screen). If the stereo is being calculated for a single user, (assuming a user´s head position or probably tracking the head) the projections can be off-axis to guarantee correctness of the parallax information (fig. 66).

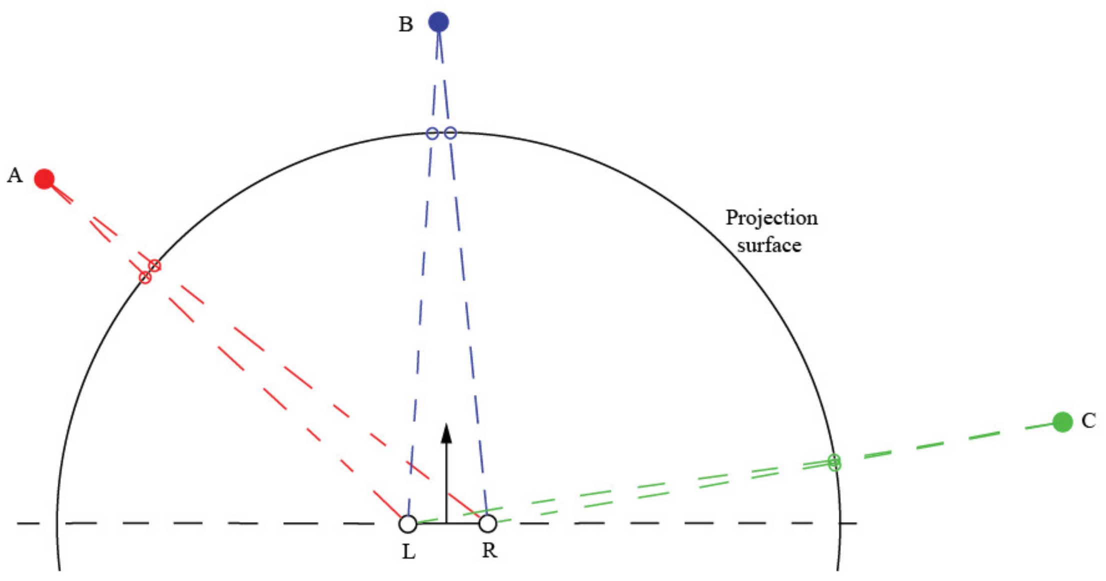


Fig 66: Curved projection for a single user.

As illustrated in the previous figure, a single viewer´s position and orientation of the head are assumed. The projections onto the screen depend on the eye´s position, but not the vertex position (the view vector remains constant). For that reason, in the example of the fig. 66 points such as *A* (which is at the same distance from the point between *L* and *R* as *B*) show less parallax disparity. In an extreme situation, if a vertex lies on any point on the line defined by *LR*, the parallax information is lost and there is not disparity since the intersections with the screen for the left and right eye are coincident. This makes sense, and it is very similar to the way humans perceive reality. The problem occurs if the user looks in another direction while keeping his head still (assuming that eyes are not tracked). These inconsistencies in the parallax information would then become obvious. Also, for multi-user systems it is not the most appropriate option since it does not offer omnistereo.

The shader that has been proposed and integrated in the system developed for this thesis offers omnistereo and is achieved by assuming that a generic user is located at the centre of the screen facing every possible direction simultaneously. What this actually means is that the position of the eyes does not stay fix, but it is determined by the vertex position. The projection curve remains still. This sacrifice makes much more sense when it is considered with the fact that eye-tracking is very rare nowadays and multi-user systems are also very common. In other words, by using this omnistereo approach, and without rendering for a specific user or without sacrificing so much realism, the scene is valid for multiple view points. In the following image a few projection examples show how this method works (see fig. 67).

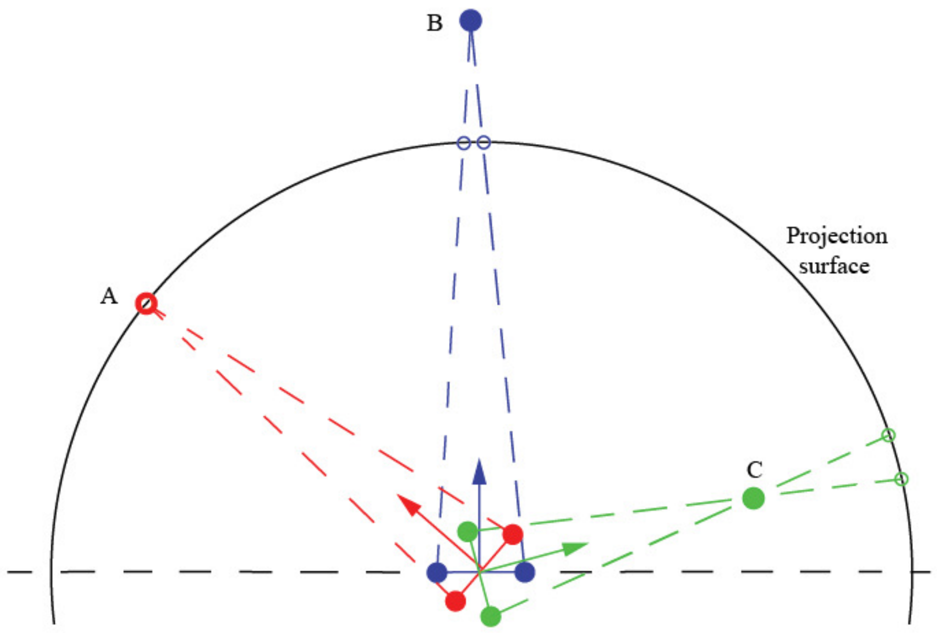


Fig 67: Curved projection with omnistereo.

Vertex B is located in the same position as vertex B of fig. 66; this is right in front of the user for the single user approach. In this case nothing changes and the same projections for left and right eye are obtained. In fact if A were at any other position at the same distance to the centre of the screen the same parallax would be achieved (which is exactly what is needed for supporting omnistereo). Zero parallax distance is also well defined, as any point located on the surface of the screen will have the same projections onto the screen for both eyes. Consequently points located within the screen area will have negative parallax (they will appear closer to the user than the screen) and those further apart will have positive parallax (creating the illusion that they are behind the screen).

This method does not require ray-tracing (calculating intersections with screen) for every pixel on the screen as a ray-tracer would do. The functionality is fully implemented in a vertex shader and consequently transforms points from world space to the clipping space (output of the shader). Consequently it can be used in real-time applications and offers much more accurate results than the multi-slice (in fact the accuracy cannot by higher since it is not an approximation of a surface), achieving a seamless omnistereo projection free of artifacts. It also supports any kind of effects since there are no additional problems with the slices. Details about the shader are explained in next chapter.

3.3.6 Synchronisation

Some immersive systems use multiple projectors either to project wider fields of view; these keep a relatively good level of brightness or obtain a higher resolution. Whatever the reason, the images projected need to be synchronised. *Genlock* [29] is not enough, since not only the frames have to change simultaneously, but also objects in the scene have to be in the same positions and orientations, with the same properties (materials, lighting, and so forth). In case of animations or video, synchronisation can become complicated because sequences are calculated in each computer independently and it is only necessary to force the update when necessary.

Consequently there has to be a part of the system dedicated to the synchronisation of the different computers that 'output display' to each projector. The solution adopted in most cases, (and this is the case for the *AVIE*), is to connect all the computers in a dedicated *LAN* network, both the computers specialised for rendering and the server. Once this is done the server can act as a master, with the computers connected to the projectors as slaves, using a determinate networking protocol.

The basic idea is that the clients only swap their back buffers when they are told to do so. For this purpose, they inform the server when they have finished writing in the back buffer (for both eyes if each computer controls the both of them) and the server waits until all the clients are ready. When this happens, the master computer informs all the clients that it has occurred, and they swap the buffers in consequence. In case two of the computers are displaying two parts of the same video or animation, the server will notify the slaves of the exact frame of the video, or the moment in the sequence, to be displayed before they render to the back buffer.

Details of the libraries used, server program and many more implementation details will be explained in the Design chapter.

Chapter 4: Design

The system developed during this Masters thesis is a cylindrical renderer for *AVIE*. Most design and implementation aspects are common to any system with similar characteristics, even in situations where the screen is not cylindrical, but spherical or any other shape. Warping, blending, calibration, synchronisation, multi-slice frustums, curved projections, omnistereo and so forth, are all key concepts common to these immersive visualisation systems for curved screens. This means that adapting the system proposed to a similar frame-work would be very straight forward. For example, *iDome* is in many ways much simpler, since it does not involve blending or synchronization. Developing a system for *AVIE* is of particular interest because it involves dealing with all of these factors.

The *AVIE* renderer has 3 main components; the calibration program, the cylindrical renderer and the server application. It is the main purpose of this chapter to present the most relevant implementation details.

4.1 The Calibration program

The calibration program brings together two fundamental aspects of the system; warping and blending. Its implementation has been conducted keeping the ease to use in mind, and offer an intuitive and user-friendly interface. The calibration program can be used for any other application that requires warping and blending, constituting a program at the level of other commercial products (i.e. *Sol7*). In fact, it introduces some new features such as more flexible control of the blending regions, different modes of defining the mesh (with or without collapse on the borders), free selection of tension for the curves of the surface, more control on the parameters of the mesh and so on.

4.1.1 Modular scheme

The scheme of the main modules of the calibration program proposed (*WarperBlender*) is as follows:

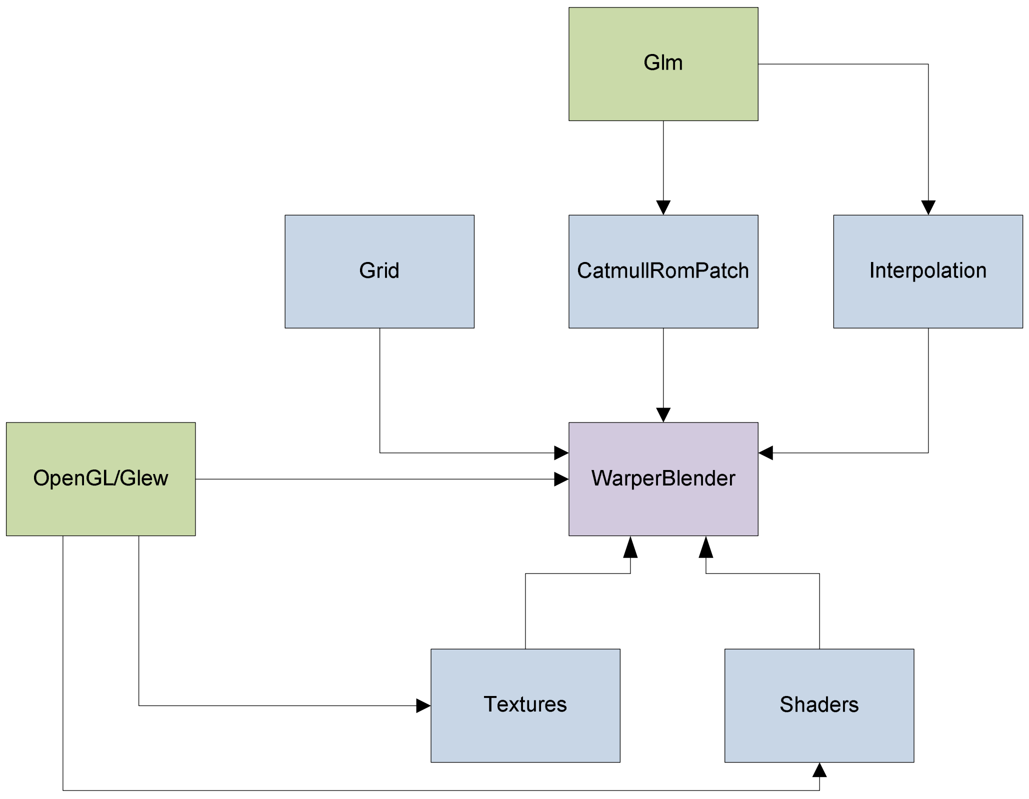


Fig 68: Modular scheme of the main components of the calibration program.

The program´s graphic library used is *OpenGL/Glew* [56][60] since it is a cross-platform open source *C/C++* extension loading library for *OpenGL*, which favours portability. Other modules such as '*Shaders*' and '*Textures*' work solely as an intermediate interface to make code clearer, but they only use *OpenGL/Glew*. '*Shaders'*, as the name indicates, contains a set of functions for compiling shaders and linking their variables. The module '*Textures'* groups functions necessary for loading textures, with the appropriate format, and so forth.

The class *Grid* is used by the calibration program to define the structure and functionality of the object *Grid*. It is initialised with a set of points structured in a matrix with dimensions corresponding to the number of horizontal/vertical control points. This class implements methods for resetting the grid, checking whether there is a control point in the area selected by the mouse, loading a previously saved grid, adding and removing control points, warping a particular point and activating or deactivating the collapse in the contours of the grid (see fig. 39). It is also of note, that when a control point is added or removed from the grid, (this implies removing a whole column or row of control points to keep the global structure, essential to calculate the patches), it needs to be recalculated as well as the whole mesh, since it is formed by the patches. It is important to keep the global aspect of the mesh. For this reason, when a control point is added, it is done in a way that the shape of the line-strips formed by the control points is respected as much as possible. It occurs in a similar way when a control point is removed. In the image below (fig. 69) an example illustrates the idea.

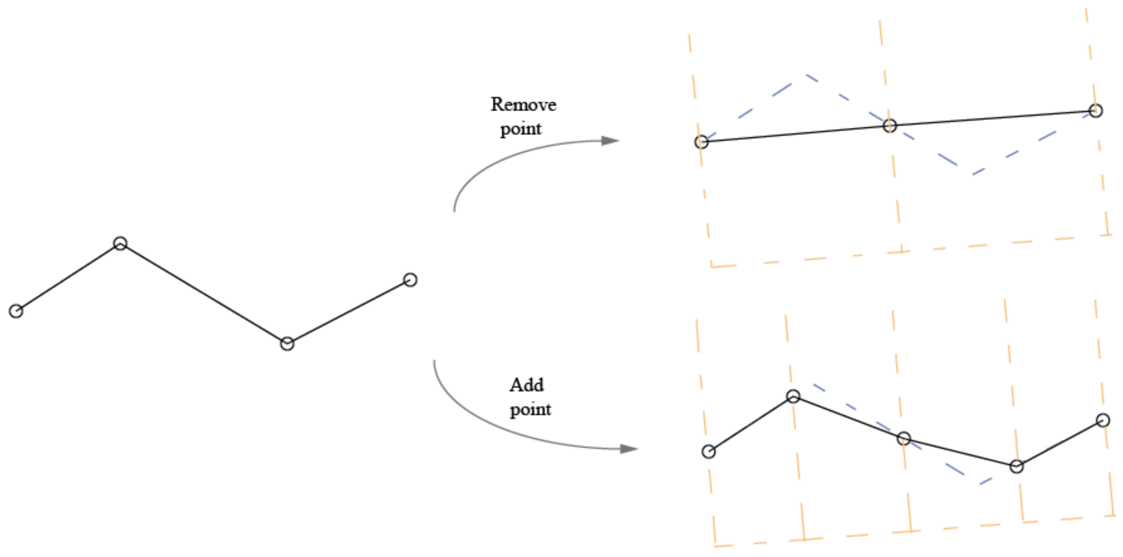


Fig 69: Example of adding/removing a control point.

In the example, a row from a mesh of 4 horizontal control points is considered. The way the functions for adding and removing points work is very similar. The distance that separates the first and last points of the strip is divided into the same number of parts as segments there will be after adding or removing the point. Once this is done, the position of the points at every increment considered is chosen according to the line that was defined before the modification.

The maximum difference occurs when the points are removed, since information regarding the shape of the strip is lost.

A matrix of patches is defined by the positions of the control points of the grid. Patches are joined to the adjacent ones as it shown in fig, 39 (with or without collapse). Each patch is defined by an instance of the class *CatmullRomPatch*, implemented from scratch for this system. This class implements methods for modifying the points, tension of the patch, and obtain a list of vertices that define the mesh that defines the surface of the patch. The number of elements in this list varies depending on the desired resolution. This means that there are multiple polygonal meshes to approximate a surface with a unique mathematical definition. Obviously, the lager the number of polygons, the better the quality obtained. As the meshes are flat (they only have two dimensions) resolution does not need to be very high. There is more information regarding this in the Results chapter. Calculating the intermediate points of the surface of the mesh is a procedure that implies many algebraic and matrix calculations, for this reason an external library has been used to aid with the task. *CamullRomPatch* class uses *Glm* (*OpenGL Mathematics*) [61] that provides many mathematic structures (i.e. vectors, matrices, etc) as well as operators defined for them (i.e. multiplication of matrices, vector operations, and so forth). Each one of the vertices of the patches is associated to one coordinate of the texture that will be mapped. Another consideration that had to be taken into account is regarding precision. The coordinates of the patches are specified in pixels, which means that there can be rounding problems (especially with low resolutions). Performing a correction each time a point is warped is necessary to ensure the polygons of each patch fit with the adjacent ones perfectly.

The Interpolation module, as its name indicates, provides functions useful for interpolating points. This is necessary for the blending part of the program (see fig. 44) owing to the fact that the blending regions can be defined as a curve passing along four control points.
There are many options, but for the program these curves have been implemented as a polynomial of grade 3. Such a polynomial is well defined by 4 parameters which are sent to the shader which applies the correction of the colour in the blending regions. These regions become darker, also taking the gamma correction into account, as explained in the previous chapter. the image on the texture. The interpolation module also uses *Glm* to make the implementation easier.

All these modules work together in the way it is illustrated in the next section, where the basic flow of execution of the program is described. There are many more details that could be described in more depth but they are not within the scope of this thesis.

4.1.2 Flow scheme

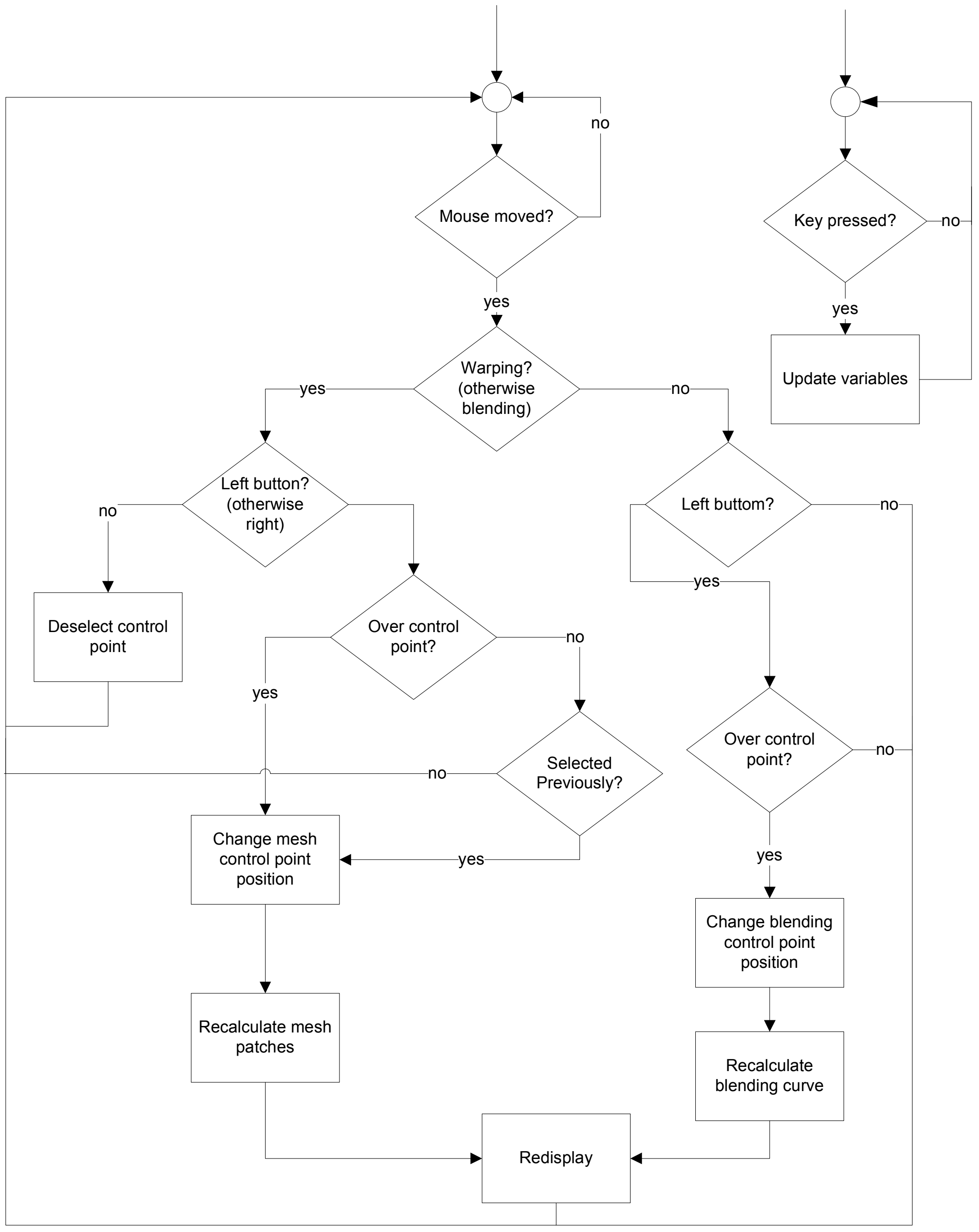


Fig 70: Basic flow scheme of the calibration program.

As in any *OpenGL* application, there are *call-back* functions associated to the input events, and for this reason the flow scheme shown above (fig. 70) has two input branches. The simplest one, shown on the right, represents the flow of the program every time a significant key is pressed. For example, when the key that controls the tension of the mesh is pressed, the variable is updated in consequence, the patches recalculated and the contents redisplayed.

However the main functionality is included in the other branch. If the mouse has been moved, the left button pressed and a control point selected (either the mouse is over an area that contains a control point or it is still selected from a previous modification) then the mouse can be moved, dragging the control point around the screen until the left button of the mouse is released. It is of interest to be able to move control point needless to have the mouse over it (only if it is selected) because in some situations it is necessary to drag the point out of the screen (it happens sometimes when two projections are to be aligned in a certain way).

The process is slightly simpler in the case that the blending mode is active. Points that control the curves defining the blending areas can be modified just by dragging and dropping them. In this case, a selection of the last point modified is also kept, but only to be able to modify particular variables of a determinate area. For example, if the last area to be modified was the right one, then the control points stay highlighted so the user knows if the key for changing the exponent of the blending function is pressed, that will be the area affected.

4.1.3 Implementation details

The calibration program expects a few parameters in its command line when it is started. In fact, if no parameters are specified the program runs without any problem, in full screen mode. However, the resolution can be specified by the user. In a system like *AVIE*, the projectors for left and right eye are mapped in a way that they display the left and right half of the computer display. For this reason, it is important that the calibration program can be started anywhere on the screen and with a customable resolution. For calibrating *AVIE*, the command line should look like the following:

```
warperBlender.exe 1400 1050 left 1
warperBlender.exe 1400 1050 right 1
```

Each eye´s calibration is carried out in two different windows (without any frames to be able to use all the pixels). It is useful to include the two command lines in a script so the configuration process can be started up easily with a single click. The third parameter (left/right) indicates that the calibration is going to be carried out on the left or right side of the screen (in *AVIE* for one projector or the other). This could be easily extended to become more flexible, in order that it may function in any other system. The last numeral indicates the number of the machine being calibrated.

Depending on the resolution, the projector that is being calibrated and the calibration itself, the program generates two files; a configuration file, "*config_AVIE_igX.txt*" and a file with the specification of the mesh "*meshL.txt*" or "*meshR.txt*" (for left or right). These files are just plain text files that will be read by the renderer in order to project the images appropriately (see next section).

One of the important features of the calibration program is that allows the control of the texture to be displayed for calibration in runtime. The first option is used for the warping, and it consists of a grid of antialiased black and white squares generated within a fragment shader. By changing the number of these squares (vertically and horizontally) the calibration process is simplified significantly because it makes easier to configure the pattern to align with the screen. A normal texture can be selected by the user as well. This is particularly useful when the blending is done for old or poor quality projectors with very different gamma values, and a specific set of colours will be necessary. The rest of the options are generated in the shader; the options include a black texture (helpful when calibrating for two eyes in the same area when one of the textures is complicating the calibration of the other), a white texture and a texture with a few colour transitions. After the texture is generated, the blending equations that were described in the Analysis chapter are applied in the corresponding area. The last detail to mention is that, obviously two blending areas can affect the same region of the texture. For example a pixel located in the top left corner can be affected by the top and left blending areas, having an additive effect.

4.1.4 Configuration files

The configuration file is generated by both left and right eye, so one will overwrite the other, but that does not imply a problem since both are identical for a correct calibration (meshes have to be perfectly aligned for both eyes). The contents of the configuration file are as in the following example:

```
#MESH_TYPE AVIE 1
#MESH_FILENAME meshL.txt meshR.txt
#CAMERA_ORIGIN 0.0 0.0 0.0
#CAMERA_ROTATION_IG1 -0
#CLIPPING_PLANES 0.1 1.0
#NUMBER_OF_SQUARES 24 16
#DIM_SQUARE_X_Y 0.225 0.225
#NUMBER_OF_SQUARES_TOTAL_AVIE 137 16
#NUMBER_OF_OVERLAPPED_SQUARES 0 2
#SLICES_CAMERA 4
#SCREEN_RADIUM 5.0
#SCREEN_HEIGHT 3.6
#USER_HEIGHT 1.6
#DOOR_WIDTH 0.85
#HALF_IPD 0.03
#RESOLUTION 1400 1050
```

All that is needed to create a sector of the cylindrical projection is specified in the information in the file, excluding information regarding warping and blending (that is located in '*meshL.txt*' and '*meshR.txt*'). Most of the parameters have an evident meaning. The "*CAMERA_ROTATION_IGX*" parameter indicates in which direction the camera is looking (let´s assume that there is a single camera for each projector). It is specified by the rotation angle from the origin (right edge of the *AVIE*´s door) to the view vector of that projection. It is specified in number of quads, (the number of quads that are displayed during the calibration and that have to be aligned to the markers located on the screen). The total number of squares of that projection is also specified in '*NUMBER_OF_SQUARES*' horizontally and vertically, as well as the total number for the whole screen '*NUMBER_OF_SQUARES_TOTAL_AVIE*' and their dimension in real world units (meters) "*DIM_SQUARE_X_Y*". The number of quads overlapped on the left and right need to be known as well in '*NUMBER_OF_OVERLAPPED_SQUARES*'. In case the multi-slice rendering is not active '*SLICES CAMERA*' does not have any effect (since only a single projection in necessary in cylindrical mode). The rest of the parameters have obvious meaning.

However, '*CAMERA_ROTATION_IGX*' is a parameter that cannot be known at the end of the calibration as with the rest of the parameters are. The reason for this is that this is an accumulated offset rotation value that depends on the other computers calibrations. Once all the individual files are generated in every computer, the value can be deduced easily by using the rest of the configuration parameters. For this reason, after calibrating, this last 'post-calibration' calculation in necessary. An additional script is also necessary for this purpose. The script is run from the server computer, (even though it might be done form any of the clients), and it accesses the configuration files of every client computer and overwrites the files with the parameter correctly calculated. The only additional information required is the path to the files.

The formula to calculate the offset angle for a given computer can be expressed as follows:

$$Offset_i = Offset_{i-1} + \frac{number\_of\_squares_{i-1}}{2} + \frac{number\_of\_squares_i}{2} - OverlapLeft_i$$

The different elements are the same as those in the configuration file. In the next image (fig. 71) an example scheme is shown.

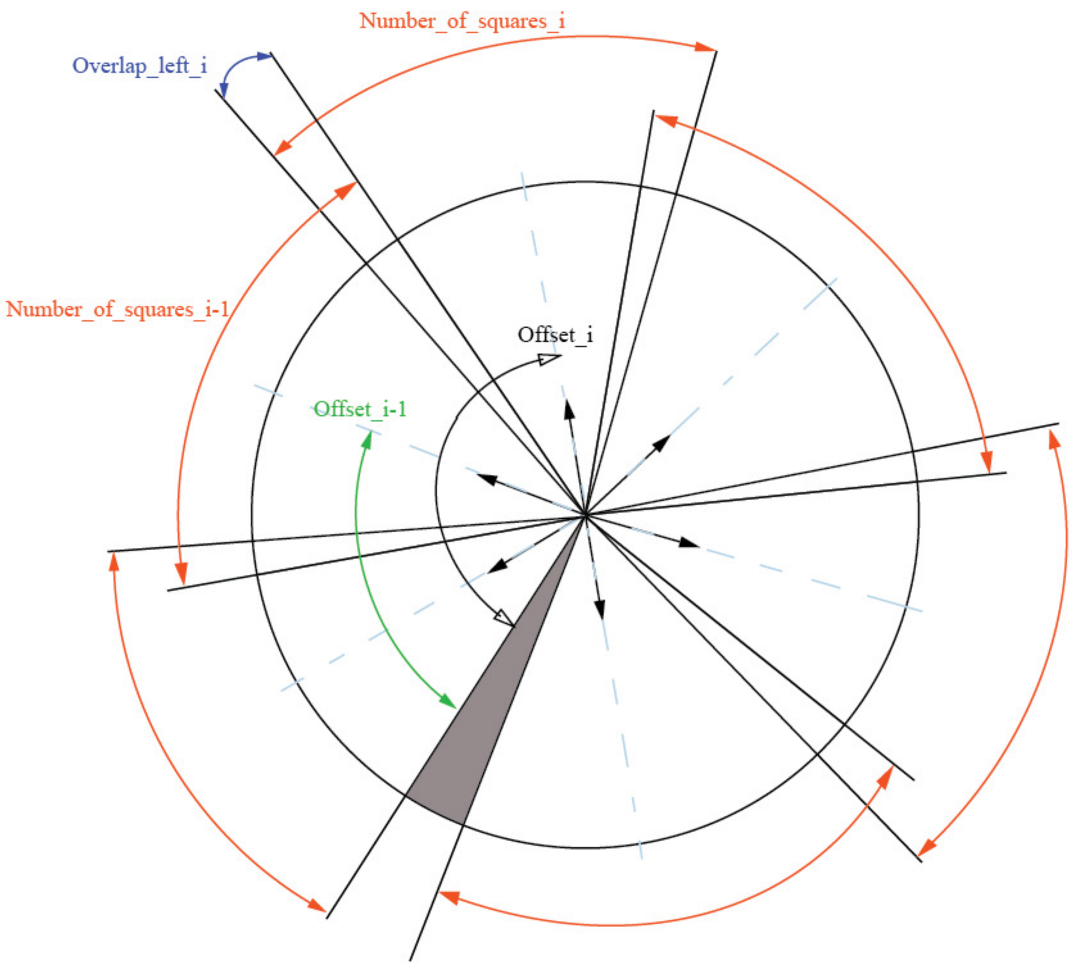


Fig 71: Process of calculating the offsets for the configuration files.

The area coloured in grey represents the door, the calibration can include the door or not. In case it is included, as no images can be shown in this area, it is blended to black, and the full angle covered would be 360º. This implies that images that will not be ~~finally~~ displayed are being rendered. On the contrary if the door is not included, the angle is less than the full circumference and no blended region is necessary for that area. The door normally is very narrow (80 cm.) and the difference is not significant, however, if the area is wider, calibrating the system taking the door into account is a better option since regions that would not be shown (blended to black) are not rendered, making it slightly more efficient. Overlapping regions are necessary, but they do not occur at the edge of the door (the beginning and end of the screen). In those cases the overlapping in the configuration file is zero.

The other files that the configuration program generates are the mesh files. These files are referenced in the configuration file by the variable '*MESH_FILENAME*' and contain all the information needed for warping and blending. As opposed to the other file, the purpose of the parameters of this file is only to warp the cylindrical projection in a way it suits the surface of the cylinder. In essence the projection that is finally displayed can be any projection (changing the parameters of the file '*config_AVIE_igX.txt*'), but the projection and blending areas will remain the same. An example of the data contained in the mesh files is shown below:

```
#CONTROL_POINTS
2 2
#NUMBER_OF_SQUARES
24 16
#VERTEXES_PER_PATCH
10 10
#TENSION_PATCHES
0.1
#BLENDING_POSITIONS_L
145 0 145 402 186 808 145 1199
#BLENDING_POSITIONS_R
789 0 575 460 789 798 552 1199
#A_COEFFICIENTS_LEFT_RIGHT
0.500000 0.500000
#BLENDING_POSITIONS_BT
38 1180
#BLENDING_LEFT
-0.574525 0.767151 -0.192626 0.151199
#BLENDING_RIGHT
4.546491 -6.833497 2.534139 0.178311
#BLENDING_BOTTOM&TOP
```

```
0.031693 0.984153
#BLENDING_EXPONENT
1.200000 1.000000 2.000000 1.000000
#GAMMA_VALUE
2.200000
#CONTROL_POINTS
1 1 80 196
1 2 96 986
2 1 822 124
2 2 623 1107
#VERTEXES_MESH
0 9 96 986 0.000000 1.0
1 9 171 994 0.111035 1.0
2 9 233 1007 0.222041 1.0
3 9 287 1021 0.333025 1.0
4 9 336 1038 0.443996 1.0
5 9 382 1054 0.554963 1.0
6 9 431 1071 0.665934 1.0
…
…
```

In this case, most of the fields of the file are self-explanatory; only a few explanations are going to be given. The parameter '*NUMBER_OF_SQUARES*' references the number of squares displayed on the texture for calibration (this has nothing to do with the definition in polygons of the mesh). This parameter is ignored by the renderer, but it is kept here in case the mesh needs to be recalibrated. The parameters '*BLENDING_POSITIONS_X*' and '*BLENDING_BOTTOM&TOP*' store the position of the control points for the blending. They are also ignored by the renderer, and kept for recalibrations. '*BLENDING_LEFT*' and '*BLENDING_RIGHT*' contain the parameters that define the blending curves (the same for '*BLENDING_TOP_BOTTOM*' but in this case is just a straight line). These parameters will be used by the renderer together with others such as '*A_COEFFICIENTS*' (controls brightness in the middle of the blending region) '*GAMMA_VALUE*' and '*BLENDING_EXPONENT*' to generate the blending texture. This implies that no texture is necessary together with the configuration files. This option avoids the use of large blending textures (for *AVIE* they would be two textures of 1400 by 1050 pixels), or the loss of precision if the blending texture is reduced. By passing the exact parameters that define the blending areas, it is ensured that the texture will be identical to the one used in calibration.

The rest is a list of control points contained in '*CONTROL_POINTS*' are used only when the mesh is loaded for recalibrations. The renderer does not need this information, only the list of vertexes that define the mesh '*VERTEXES_MESH*'. An example vertex is:

```
0 9 96 986 0.000000 1.0
```

The first two parameters are the index of the vertex that define the polygons of the mesh (defined from bottom to top and left to right). The following two parameters are the coordinates of the pixel and the last two are the coordinates of the texture to be warped. Coordinates of the pixel could be normalised [62] and the indices could be obviated (if an order is assumed). They have been kept in this way for simplicity.

4.2 *Ogre3D AVIE* renderer

The renderer system developed is specialised in a AVIE system, and consequently is able to generate 360º panoramic cylindrical scenes. It can be used to display images, videos and render synthetic scenes. There are many possibilities that can be implemented for such a system. This renderer is going to be used for an application that will be active during the next few years in *Melbourne museum*, in Victoria (Australia). The design process and all the characteristics that are included in the final product have been determined by the requirements of the application. The system was required to display panoramic 3D videos, as well as normal 3D videos and images. It also was required to display text. Objects can be defined in screen space and can be moved and oriented freely. These objects have to face the user, similarly to the way a *billboard* object works. Other characteristics have been implemented with more generic purposes or for future extensions of the system. Materials, *Phong´*s lighting, *bump-mapping* and so forth are a few examples. The system works both with a multi-slice projection approach and a fully shader-based cylindrical projection approach (since this offers more quality it is the option that will be normally in use for the museum´s application). In addition, efficiency was also an issue to consider, while keeping the whole system synchronised at every level (refresh of the screen and frames of video and animations).

The next section presents a modular scheme of the application that will be explained in-depth, together with the libraries involved.

4.2.1 Modular scheme

The scheme of main modules of the render system is as follows:

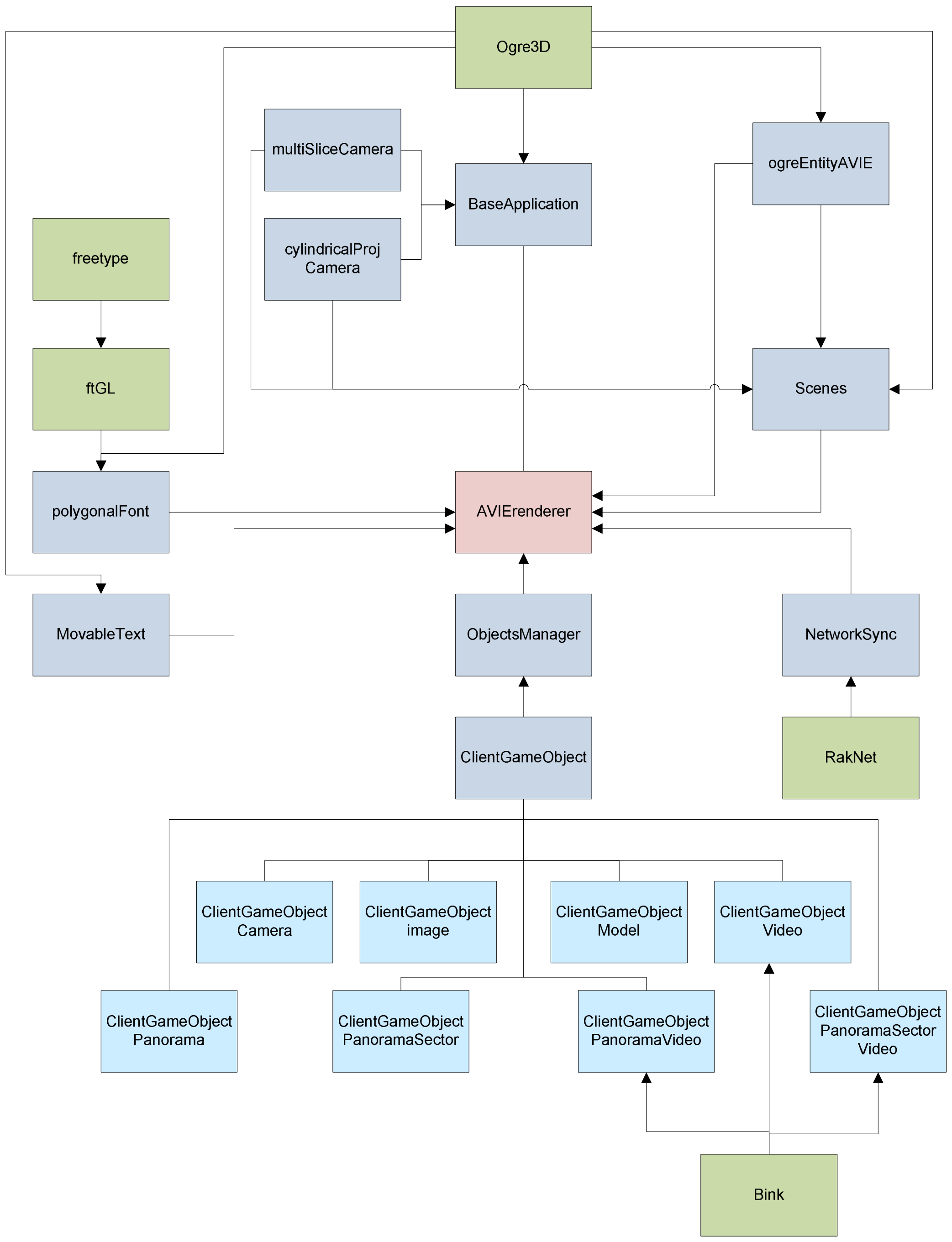


Fig 72: Modular scheme of the main components of the renderer program.

In the next sections, details and methods of the classes implemented are commented on, even though there are many more that can be consulted directly from the source code of the renderer only the most relevant are discussed.

As it can be deduced, the main class is *AVIErenderer* that extends other classes and uses functions from other modules. In fact, *BaseApplication* implements some interfaces of some classes of the 3D engine, *Ogre3D*. *Ogre* stands for *Object-Oriented Graphics rendering Engine*. It is a scene-oriented, flexible 3D rendering engine written in *C++*. This class library abstracts the details of using underlying system libraries like *Direct3D* or *OpenGL*, and provides an interface based on world objects and other high level classes. As it is a completely open source render system it provides absolute freedom and control to the programmer, and can act on any point of the rendering pipeline.

Ogre3D is a large library full of resources and it very easy to include new features since there are many plug-ins available developed by others in the community, that support the library. It is not the purpose of this thesis to go through all the features and details of implementation of *Ogre3D*, for further information a programming manual in *Ogre3D* can be consulted [63] as well as well as the many tutorials and examples available on the net.

Just to illustrate the most important core objects of the library, the following graph of dependencies is provided.

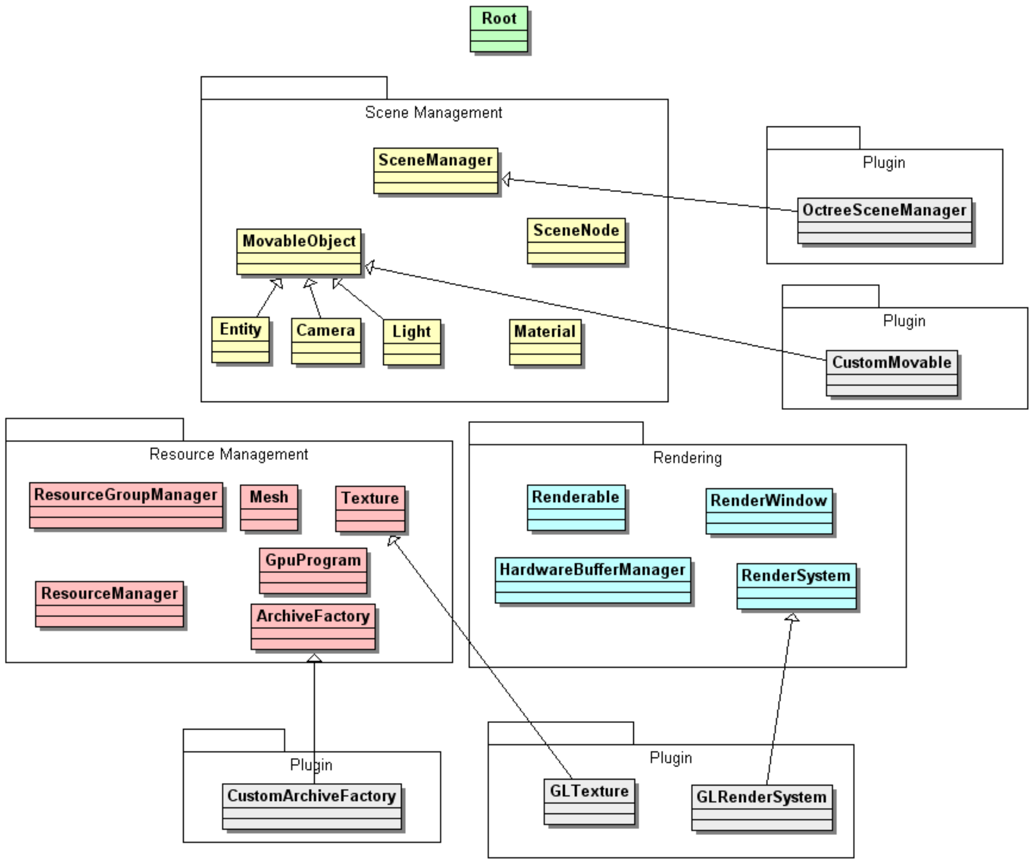


Fig 73: Dependencies diagram of Ogre3D´s core objects (Ogre Manual, [64]).

The class *BaseApplication* can be seen as a 'standard' *Ogre3D* application that implements methods and interfaces necessary for an example program, such as the interface, frame listeners, size of the window, viewports and so on. *AVIErenderer* extends this class and re-implements some of the methods with the code necessary for a cylindrical slice renderer.

PolygonalFont and *MovableText* provide the possibility of using polygonal text (constructed as a mesh) and or plain text as a texture. These two classes are modified versions for a cylindrical renderer of two existing *Ogre* plug-ins.

OgreEntityAVIE extents the class *OgreEntity*. An 'entity' in *Ogre* is any object that is displayable on the screen. However, when the cylindrical renderer (using the cylindrical shader) is active, the way materials are applied is slightly different, in order to use the shader appropriately. *OgreEntityAVIE* implements a method that facilitates the task.

Ogre3D only allows the programmer to define perspective and orthogonal cameras. For this reason it has been necessary to create a class 'camera' for each type of projection; *multiSliceCamera* and *cylindricalProjCamera*. In this chapter there is a section regarding them.

The *NetworkSyncClassRakNet* class is the class used for synchronisation. It utilizes *Raknet*, a cross platform, open source, networking engine. An instance object of this class is necessary for the synchronisation to the rest of the computers. The behaviour, in synchronisation terms, of the program is shown in the next section (flow scheme), and it is possible thanks to the object instanced from this class.

The *ClientGameObject* class is accessed by the *ObjectsManager* class, every time an object is added or removed from the scene. It creates the data structure needed, attaches it to a node and to the *Ogre´*s scene graph, as well as the linkage to the corresponding pointers in the *ClientGameObject* objects (necessary for moving, scaling, and so forth). There is a section in this chapter that discusses the *ClientGameObject* class and its subclasses.

4.2.2 Flow scheme

The following flow scheme shows a high level trace of the program´s behaviour, especially that which regards to synchronization.

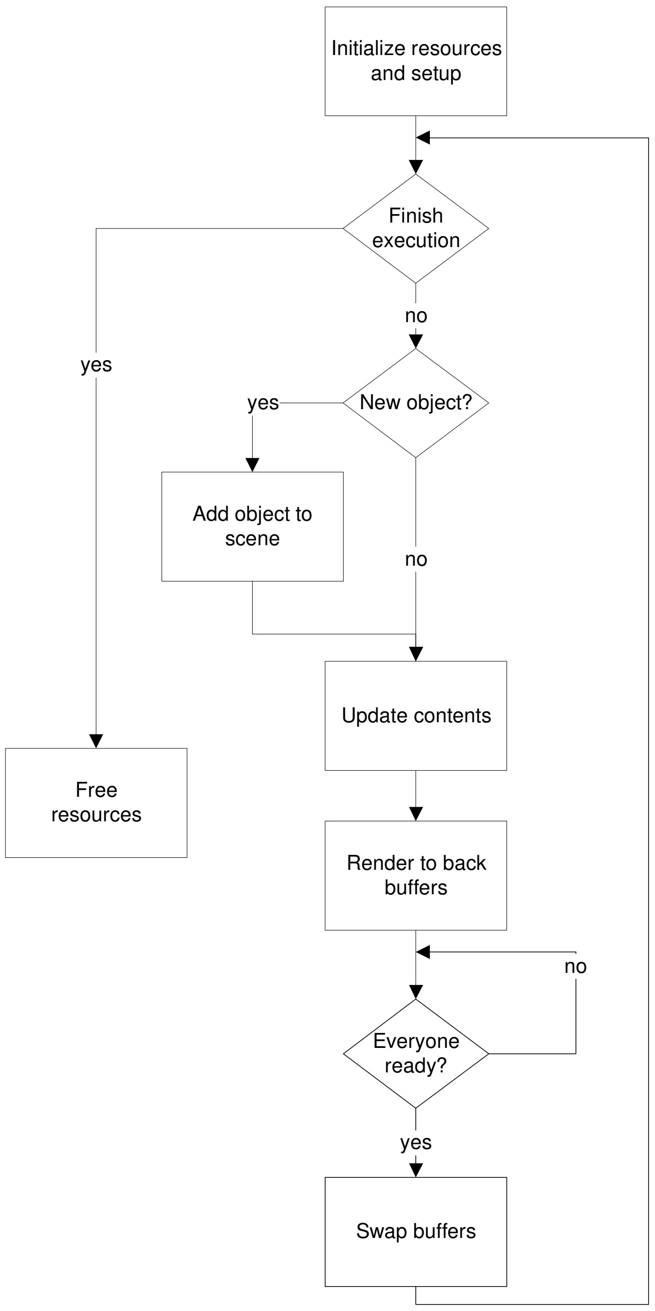


Fig 74: Basic flow scheme of the renderer program illustrating how synchronization works.

The program begins reading the configuration files to obtain the parameters of the warping mesh, blending areas and view volume. Also, all the necessary resources are loaded to memory (the resources that are needed can be specified in the *resources.cfg* file or loaded individually). Once this is done, the scene is composed by the server by adding, placing, scaling and orienting every necessary object. This can actually be done any time, and not only at the beginning of the execution. For every frame to be rendered, synchronisation at frame level is essential. Consequently the scene is rendered to a back buffer as usual, but the buffers are not swapped automatically. When the execution reaches that point, a message is sent to the server informing it that the new image is ready to be displayed. The front buffer image is kept until every computer in the network has done the same and when this occurs the server sends a message back to the client computers to inform them that all the images are ready, so they can swap the buffers and prepare the next frame. The automatic swapping of buffers in *Ogre3D* is disabled for this system and methods that do the swap 'manually' are implemented in the camera classes. When the server sends a message to finish the execution, the program frees all the resources that have been loaded and the execution finishes.

4.2.3 Game objects

Game objects work as the 'glue' between the server program and the renderer. Every time the server adds an object, the server keeps a pointer to that object so it can modify important features such as the material, change the orientation or position. Most of the actions that can be performed by the objects are specific to the type, or at least, they are performed in a very different way. For this reason it has been believed to be convenient to split the *ClientGameObject* class into several subclasses.

Probably the simplest of these subclasses are the *ClientGameObjectCamera* and *ClientGameObjectModel*. When a camera is added, the server takes control of the cylindrical camera and can move around the scene freely. *ClientGameObjectModel* adds a polygonal model to the scene.

The *ClientGameObjectImage* displays an image mapped onto a quad. The server can change locate and orientate the image freely. Materials are also up to the user. In some situations 3D images are necessary. This means that not only the quad will appear in a 3D location in the scene, but the image displayed within that quad will also be different for left and right eye (otherwise it would look like a normal flat picture pasted onto a square). The image can be changed anytime and the display mode can also be swapped between 3D and normal mode.

The *ClientGameObjectVideo* is very similar to *ClientGameObjectImage*. The main difference is that the texture changes within determinate time frames. This is a complicated issue since the synchronisation is more difficult in videos than, for example, in animation sequences for meshes (which is also supported). To be able to access a determinate frame of the video sequence, videos are encoded in *bik* format, and loaded with the library *Bink* [65]. *Bink* is a completely self-contained, simple and powerful video codec that simplifies this process.

There are also *ClientGameObjectsPanorama* (images and videos) that are useful to display an image in the background of the screen. Instead of mapping an image onto a square, in this case a cylinder is created. The radius of this cylinder is controlled by the programmer. For example, if a 3D video needs to be displayed, an additional disparity between what the left and right eye see is not desirable and the cylinder has to be created at the distance of parallax zero (touching the screen). In this case, if the panorama is supposed to be located in the background, the option that avoids that this object writes in the depth buffer is so that any object (behind or in front of the screen) can still be seen by the user. *ClientGameObjectsPanoramaSector* is used when high definition videos are displayed in the background. It would be very time consuming to decode the whole video if only one sixth of the video was going to be rendered for each computer. The solution is to divide the videos in 6 parts (6 parts of 60º each) and create a cylindrical section instead of the whole cylinder, and map only the corresponding part. As a function to obtain 3D coordinates from screen coordinates was implemented as well, this process is relatively straight forward.

4.2.4 Cameras' implementations

Two different camera implementations are implemented in two classes. Both are interchangeable, but their properties and the way they are implemented are different. As a result they render to a texture the scene for left and right eyes, for every frame.

4.2.4.1 Multi-slice camera

The *multiSliceCamera* class implements a cylindrical projection by approximating the surface with off-axis planar perspectives (see fig. 61). All the calculations regarding off-axis projections, angles, viewports and so on are completely abstracted from the programmer. When an object of this class is created, it automatically generates two output textures (for each eye) with the projected images. The profile of the method that sets the parameters of the projection is as follows:

```
void multiSliceCameraAVIE::setParameters(
      int nCameras,
      float angleCoveredTotal,
      float nearClippingPlane, float farClippingPlane,
      Ogre::Vector3 position,
      Ogre::Quaternion orientation,
      float aspectRatioTotal,
      float screenRadius,
      float screenHeight,
      float userHeight,
      float halfIPD,
      Ogre::SceneManager* _mSceneMgr,
      int resolutionX, int resolutionY
);
```

As shown in the profile, all the parameters are extracted or derived from the parameters in the configuration file. Also, a pointer to the scene manager of the 3D scene is necessary for creating the Ogre cameras (off-axis perspective cameras). The scene that displays the warped meshes is a different one, associated with another scene manager (that will display an orthographic projection of the scene with two warped meshes, one next to the other).

No warping of the textures is performed if it is not related with the cylindrical projections themselves, but with the position of the projector respect to the screen. Warping has been isolated from the camera class because the system also offers the possibility of displaying the image without any warping applied. However, as blending is a post-processing of the texture it occurs within the camera class. The profile of the function that sets the parameters is displayed below.

```
void cylindricalProjCameraAVIE::setBlendingParameters(
      float _shadeTopL,float _shadeBottomL,
      float _exponentBottomL, float _exponentTopL,
      float _coefficientsLeftL1,float _coefficientsLeftL2,
      float _coefficientsLeftL3,float _coefficientsLeftL4,
      float _coefficientsRightL1, float _coefficientsRightL2,
      float _coefficientsRightL3, float _coefficientsRightL4,
      float _exponentLeftL,float _exponentRightL,
      float _aLeftL,float _aRightL,
      float _gammaL,
      float _shadeTopR, float _shadeBottomR,
      float _exponentBottomR, float _exponentTopR,
      float _coefficientsLeftR1, float _coefficientsLeftR2,
      float _coefficientsLeftR3,float _coefficientsLeftR4,
      float _coefficientsRightR1, float _coefficientsRightR2,
      float _coefficientsRightR3, float _coefficientsRightR4,
      float _exponentLeftR, float _exponentRightR,
      float _aLeftR,float _aRightR,
      float _gammaR
);
```

In the same way than the function for defining the parameters of the projection, the parameters of the blending are extracted from the configuration files (in this case from the mesh files). If no blending parameters are defined or the blending is deactivated manually by the user, the projections do include the blended regions.

Another interesting function implemented by the cameras is the possibility of adding *billboard* objects and 'follower' objects to the scene. This is necessary because the way an *Ogre billboard* is defined it requires a specific orientation from a camera (defined as a quaternion). These cameras can have a very wide angle, so no specific orientation is defined (the cameras can be looking in every direction). For this reason, if the *Ogre* billboards are used, the object would face a different direction for each slice. A 'follower' is an object that follows every movement of the camera (it could be useful for implementing a *head-up display*, for example). Not only methods for adding and removing these elements are necessary, but also the fucntions that update their positions or orientations. The camera keeps a list of elements and every time it is asked to update them, the changes will apply (if the camera has moved). There is more information about the billboards in another section of this chapter.

To update the position of the cylindrical camera, a master camera is defined. This is the camera that would be in the centre of the set of slices and an *Ogre* camera can be used for that purpose. Just by using the method *updateCameraPositionOrientation* with the values that can be extracted from the master camera, all the off-axis cameras for every slice are updated in consequence. It is easy to determine the orientation of each one by finding the offset angle in respect to the centre of the one that is in the left (or right) extreme of the set of cameras. The formula to calculate this offset for an even number of cameras (fig. 75) is:

$$OffsetAngleLeft = \left\lfloor \frac{numberOfCameras}{2} \right\rfloor \frac{angleCoveredTotal}{numberOfCameras} - \frac{angleCoveredPerCamea}{2}$$

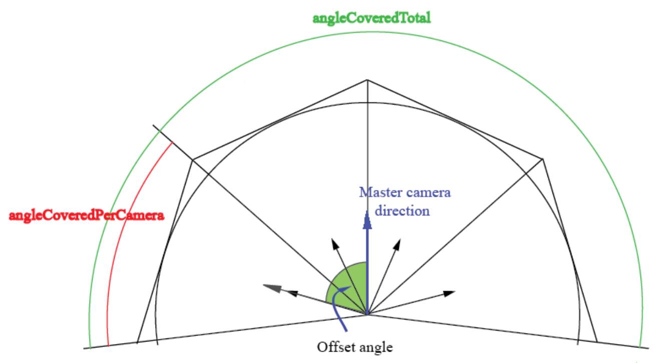


Fig 75: Offset calculation with an even number of slices.

And if the number of cameras (fig. 76) is odd:

$$OffsetAngleLeft = \left\lfloor \frac{numberOfCameras}{2} \right\rfloor \frac{angleCoveredTotal}{numberOfCameras}$$

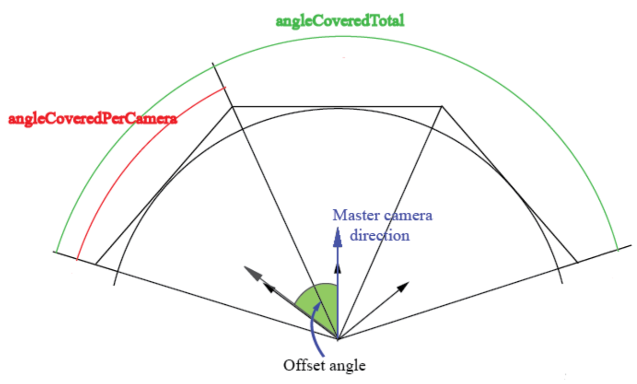


Fig 76: Offset calculation with an odd number of slices.

The offset angle is calculated for the centre of the area corresponding to a slice. The images are shown for a non-stereoscopic projection for simplicity, but having offset projections instead of symmetric projections does not affect the calculation.

4.2.4.2 Shader-based camera

The shader-based camera implements very similar methods to the multi-slice camera. The interfaces of the functions are the same, and the programmer initialises the parameters in the same way. For both, the result is the 3D scene rendered to two textures (for each one of the eyes).

The main difference is that the projections are not performed by planar projections. Actually, a single camera with symmetric view volume is defined. It works only as a way of splitting the space into regions. For example, one of the six computers needs to render 65º of the scene, so internally to the class there is a perspective projection defined that remains useful to convert from world coordinates to camera space (even though the projection matrix will not be used) for anything else than performing the *view frustum culling* of objects. It is important to note, that for the stereo mode, the two frustums of the cameras are exactly the same (again, they work just as a way of dividing the vertices of the scene in regions). As projections of more than

180º cannot be defined in *Ogre3D*, rendering a scene of 360º normally 3 projections are needed. Another option would be to deactivate the view frustum culling option, (process that clips objects that are completely out of the view frustum so none of their vertices are sent to the graphics pipeline), and use a single 360º projection (even though this option is not recommended for reasons of efficiency).

Each of the cameras has their own parameter that is sent to the vertex shader which indicates if it is the camera for the left or right eye. It is within the shader that the projection onto the curved screen is conducted for the left or right eye, also considering the specific position of the particular vertex. Let us also illustrate with an example how the method works in extreme situations. In fig. 77 there is a point which is very close to the border between two view areas (probably two different computers). For the projections associated with the area 1, the left eye is able to see the point, but not the right eye, since it intersects outside of the region and will be clipped. However, in the projections associated with area 2 the opposite occurs. The projection for the right eye includes the point but not the left one. The result is correct, since the point is only projected once for left and right eyes. If the point is on the border, the same process is repeated.

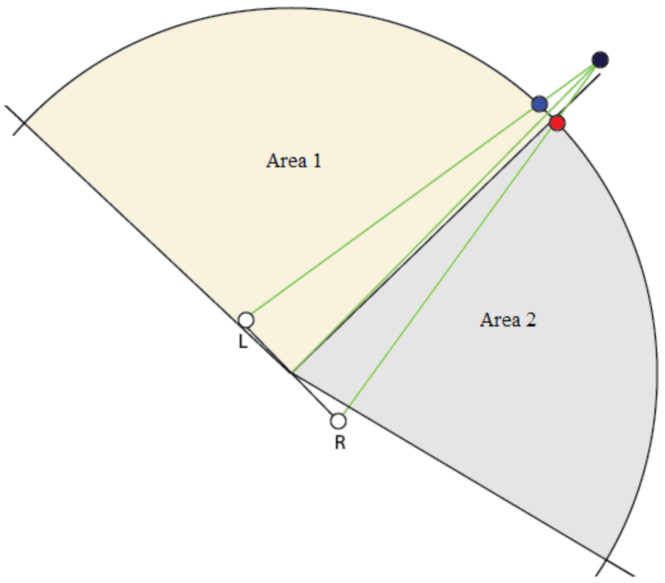


Fig 77: Cylindrical shader´s robustness against extreme situations.

However, an extension had to be made, even though this situation very rarely happens (large interocular distances raise the risk). If, instead of being a point is a whole tiny mesh (let´s say a little cube), it may occur that the point is clipped in the view frustum culling stage since it is not contained by area 2. In this case, the right projection of the point would be lost. In fact, as there is an overlapping area between the view frustums, this problem could be imperceptible since the border area for one of the frustums is not the same for the adjacent one and vice versa, but for correctness it has been fixed. The first option would be to switch off the view frustum culling in *Ogre3D*. This is not realistic, since scenes with a high number of polygons would make the execution impractical. The solution adopted by the system then, is to included a slightly higher horizontal field of view (according to the interocular distance) so this situation never occurs.

Once the intersection with the screen has been found, the next step for the vertex shader is to convert the point from Cartesian space to a cylindrical system (see appendix 2) to be able to perform the conversion to Clip coordinates. Clip space defines a cubic view volume in order to simplify the next step of the graphic pipeline. The coordinates of this cube are defined in the intervals $-w \leq x \leq w$, $-w \leq y \leq w$, and $-w \leq z \leq w$, in *OpenGL* ($0 \leq z \leq w$ in *DirectX*), being w the homogeneous coordinate [66] of x ,y, z. By having the truncated perspective pyramid deformed to a cube, it is immediately determinable which objects are covering others, just like in an orthographic projection (see fig 78).

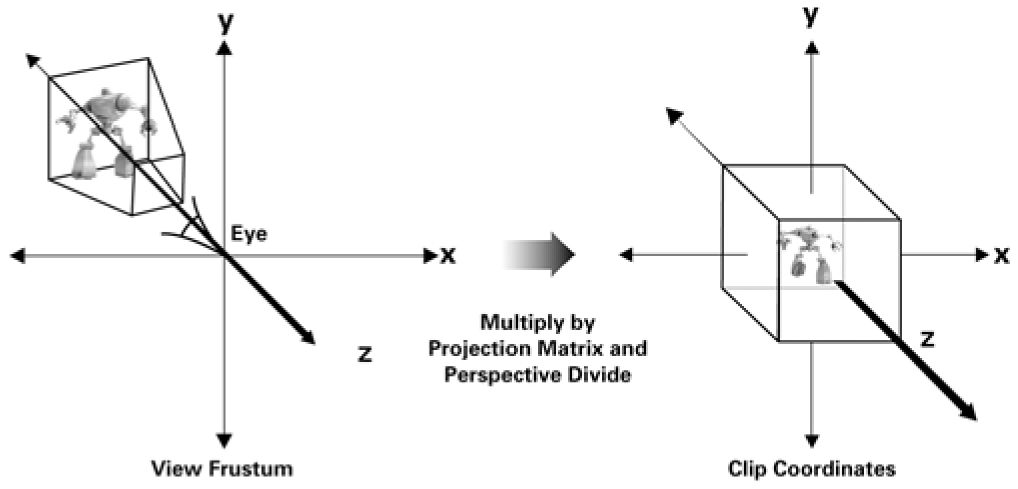


Fig 78: Transformation from World coordinates to Clip Coordinates (Kilgard, 2003 [67]).

The only difference in this case is that it is not a truncated pyramid, since it is a curved space in x and z dimensions (vertical cylindrical perspectives still perform a linear perspective transformation vertically, coordinate y).

In the image below (fig. 79), it easy to see that knowing the total angle of the cylindrical view volume (from above since, the vertical aperture is determined by a normal perspective projection) and the angle that the point forms with the edge of the volume (fig. 79a) the coordinate x in clip space can be easily obtained (fig. 79b). The same applies for the z coordinate of the clipping space, which this time is obtained by calculating the relative distance from the near clipping curved surface of the view volume and the far surface. In this case these surfaces are not planes so they must be determined by their radius.

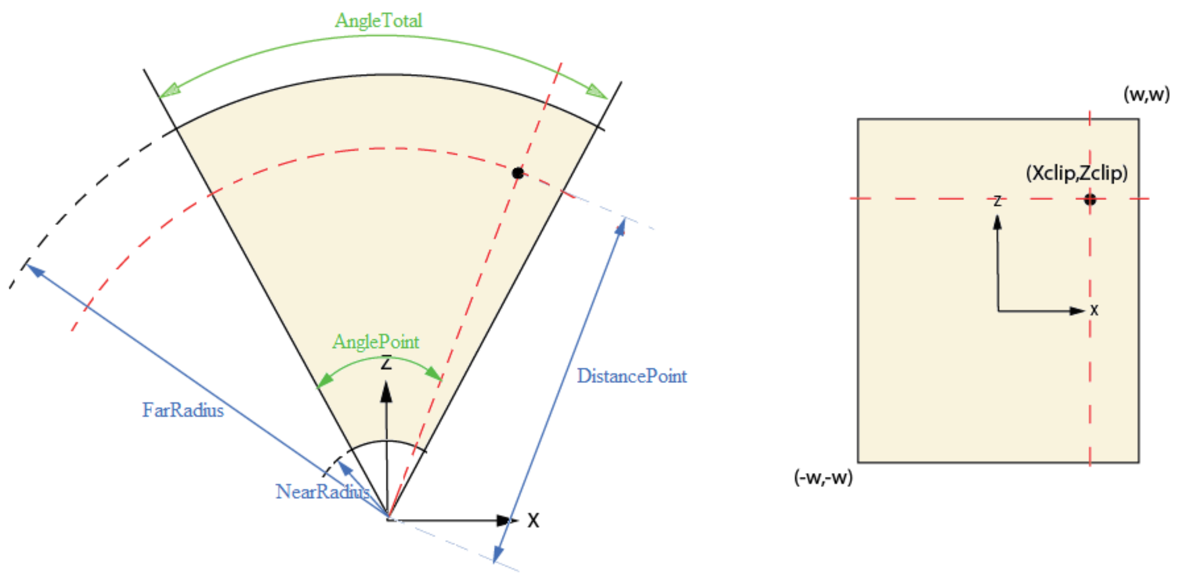


Fig 79: Transformation from curved view volume on the left (a) to the Clip space on the right (b)

The value w is the homogeneous coordinate and it is determined by the transformation matrix. As we are calculating it ‘by hand’, it is sufficient to set it to 1. This means that coordinates output of the vertex shader have to be defined in the interval [-1,1] for the 3 dimensions.

One last problem that had to be solved is described as follows. Texture mapping requires the w value to interpolate the texture coordinates along the polygons. Otherwise an affine texture mapping needs to be performed, affine texture mapping interpolates linearly across the screen, which conduces to poor results when the polygons are observed at an angle (the polygons look bent). The interpolation has to be done in function of the distance to the observer, and for this reason, before finishing the execution of the vertex shader the output vertex is multiplied by the distance to the observer. By doing this, a correct texture mapping is observed (see comparison in fig. 80). The output point then becomes ($x{\cdot}dist, y{\cdot}dist, z{\cdot}dist, dist$).

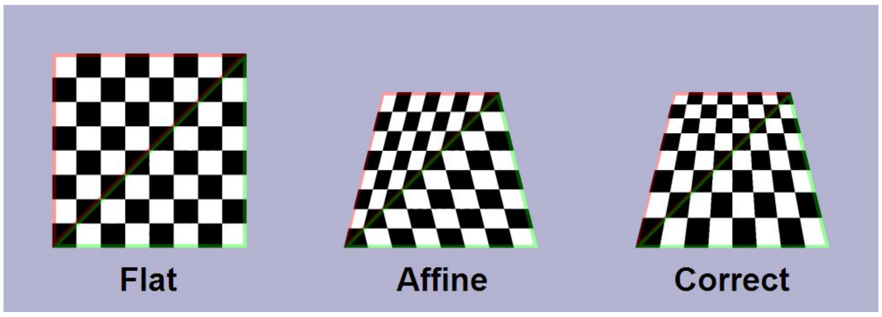


Fig 80: Texture mapping techniques, affine and correct (*Wikipedia* [68]).

4.2.5 Other features implementation

4.2.5.1 Screen coordinates to 3D space converter.

In order to facilitate the task of loading scenes, a function that changes from screen coordinates to 3D space has been implemented. The profile is specified below:

```
Ogre::Vector3 get3DcoordsFromScreenCoords(Ogre::Vector2 screenCoords,
                                          bool normalized);
```

The programmer can request a change of coordinates providing the input data in absolute or normalised coordinates (by specifying it in the Boolean variable 'normalized'). For example if the user asks for the point (0,0) and (1,1) in normalized coordinates, the vectors obtained would locate the objects in the lower part of the right edge of the door (where the screen starts) and the upper part of the left edge of the door (where the screen finishes). This is very useful since the objects can be relocated from the specified position and also moved back and forward (to locate them behind or in front of the screen).

This conversion function is also very useful for implementing the functionality for creating panoramic images/video by sectors. As aforementioned, especially high-definition panoramic videos, require the decodification of a lot of data per each frame. This does not make sense, since only a sixth part (approximately) of the whole video is going to be displayed by each computer. The solution consists in fragmenting the video in portions, normally six sectors of 90 degrees pointing in directions separated by 60º each. Using 90 degrees may be thought to be too great, since not more that 70 degrees are displayed including overlapping regions. By doing so, the pre-process of the video can be completely calibration-independent; it is enough to calculate the normalised coordinates of the fragmented videos and convert the coordinates to screen space by using the function implemented.

4.2.5.2 Billboards.

Billboards will be used together with the function to obtain the coordinates from the screen space for the museum application, but not necessarily. Very often it is of interest to have objects facing the user or a specific location within the screen.

The *Ogre3D´*s billboard function could not be used because of the reasons previously mentioned. The problem with panoramic projections is that no specific view direction vector is defined if there is no head tracking. A vector in space specifies a direction, but not an orientation, unlike a quaternion does. A quaternion is a vector with a defined rotation around the axis. A vector does not define this rotation and consequently infinite positions around the direction defined are possible. For this reason, an important assumption needed to be made enable a definition of *billboard* objects in a panoramic framework. This assumption is determining a standard Up vector for all the billboard objects; the vertical axis. This makes sense because in the *AVIE* system for the museum, as it is assumed that users will have their heads vertically.

If the camera moves, all the billboards have to be updated, and for this purpose a method in the camera classes were implemented.

4.2.5.3 *Bump mapping, Phong* shading, and other effects.

By using the cylindrical projection approach, hazards of percetible discontinuities when applying shader based effects disappear, as opposed to what happens with the multi-slice approach. This leaves the door open to many interesting effects that can now be applied very easily.

To demonstrate this, *Bump mapping* and *Phong* lighting (per pixel illumination) techniques were implemented. The adaptation was very straightforward, since the changes that were necessary to introduce were minimal. For example, bump mapping uses tangential space to calculate illumination but that is not affected by the cylindrical projection since all the necessary vectors can be defined in object space. In the Results chapter there are some screen captures of these techniques.

4.3 Server

The server not only works for synchronising the computers, but it is also very important to distribute executable files and media among every computer. If new media (video, images, meshes, fonts, materials and so on) is to be used, every single computer on the net should have the files locally to maximize performance. For that reason the server implements an option that distributes a folder called '*media*' with all the new contents.

If a new version of the program is developed, the folder with the necessary executable files has to be distributed as well. The folder is called *runtime* and includes the executable file, all the *dlls*, shaders and configuration files (specific to each computer).

Once this is done, the system is ready to run the renderer in *AVIE*, Execution starts (not necessarily all the computers have to start at the same time, this can be configured as well) and from the exact moment that a computer joins the network, the server will take it into account together with the rest of the computers for synchronisation purposes. Since all the computers are running frame-synchronised, if the server sends an '*add object*' message, it will be added to every single scene tree of the computers in the network. The same applies to eliminate, modify properties, position or orientation of the objects.

If a scene is being modified manually by an administrator in the server side, it is useful that the complete scene can be visualised at once. For this reason, the server can also start an *Ogre3D* window that displays the whole cylindrical screen in one window, without warping or blending.

Chapter 5: Results

The main purpose of this chapter is to discuss the most important features of the system developed in order to justify that the objectives proposed have been achieved. The different parts of the system can be seen as separate modules of the same application, but also as individuals. For this reason, this chapter´s structure contemplates specific parts of the system separately and together.

5.1 Calibration program

The calibration program has been developed completely from scratch and offers similar features to some other commercial products used for the same purpose, improving some aspects, offering more control and information.

Users that need to calibrate a system can do it very intuitively thanks to the friendly interface and the level of feedback about the process that the user receives. This can also be configured by selecting the verbosity level. For example, the user can see just the warped texture, control points, the grid that defines the mesh, patches, as well as information about the variables and mouse position if the user requires it. By using the mathematical model for the surface that was employed, the curvature of the warping surface can be varied smoothly. This is a rare feature in calibration programs, since most only offer the possibility of selecting a linear or curved mesh. By giving the user the option to change the tension of the mesh, the calibration process becomes easier, especially for very irregular surfaces.

However, not only the surface´s tension can be easily configured, but also the definition (number of polygons) and the shape in the contours (collapsed/not collapsed mode). This last characteristic is very interesting since it is sometimes easier to calibrate a multi-projector system if the mesh is built in a way that the boundaries are 'prepared' to be joined to the adjacent ones. However, the contrary also applies. Normally single projector systems are difficult to calibrate at the borders and the 'spiky' (see fig. 81) appearance of the mesh cannot be avoided. By allowing the user to select between these two modes, the configurability of the system is improved.

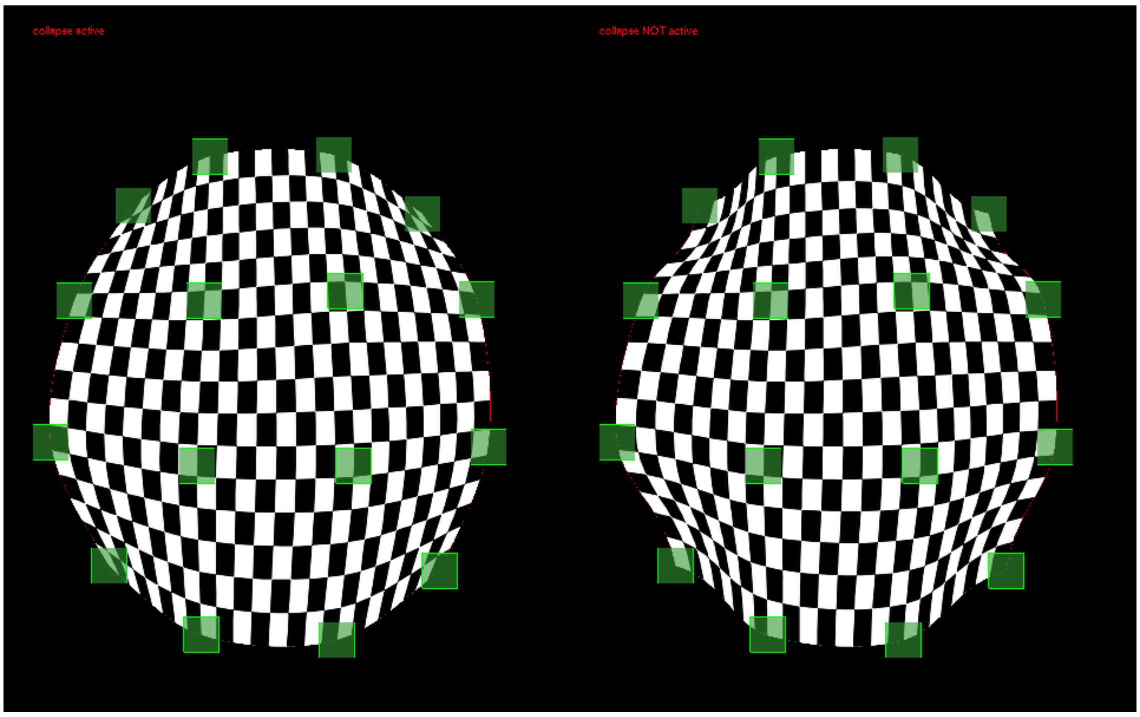


Fig 81: Calibration for two viewports (left and right eye). The collapsed mode, on the left produces softer contours than the not collapsed mode (on the right) for the exact same control points.

The global appearance of the mesh is conserved when adding or removing control points, which is a very useful feature since sometimes it is interesting to add a row or column of control points to have more control over a region of the screen (maybe because is more irregular or the projector is at an awkward angle). Users would find it frustrating if the mesh changed significantly in the middle of the process just because control points were added. The mathematical model used forces the vertices of the patches that compose the whole surface to pass through the control points (contrary to what a *Bezier* surface would do, for example). This is very useful since it allows the user to associate specific parts of the texture to a determinate region of the screen by just dragging and dropping over that area, while keeping local control (further regions of the mesh are not affected) and continuity of the curves. This property may also be used for future extensions such as an automatic calibration system based on light sensors as described in the Analysis chapter.

The blending option of the calibration program also offers more control for the user. Unlike many other calibration programs, the blending can be defined by a curve controlled by 4 points that can be moved together or individually. This curve defines the boundary of the blending areas (see fig. 82). The blending function within these areas can also be highly customised, from linear to highly curved. Gamma corrections are also taken into account.

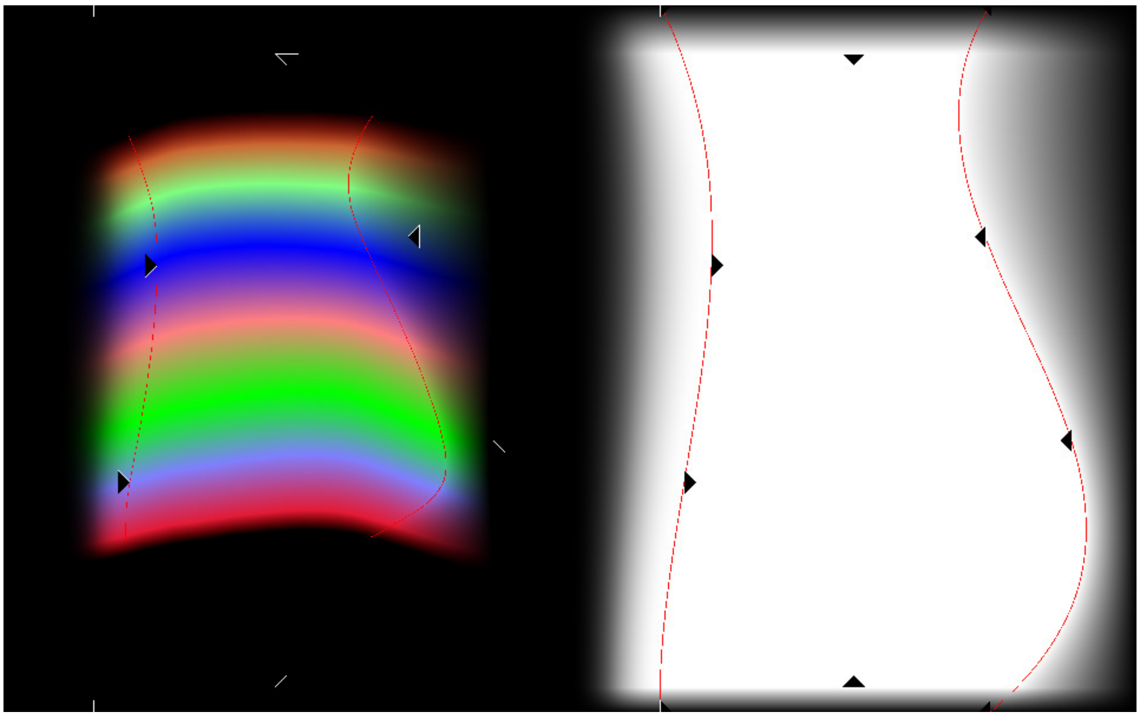


Fig 82: Colour pattern texture applied to a warping mesh on the left and the same blending mask shown on the right. Blending boundaries are highlighted in red.

The output of the program is prepared to be used in an *AVIE* system, but any other application may benefit from the calibration easily, since the output files are just plain text containing the mesh specification as well as the blending parameters (see Design chapter).

The calibration program uses *OpenGL*, which favours portability to other platforms (i.e. *Windows, Linux, MacOS* and so forth). It also offers the possibility of incorporating new features easily.

5.2 *Ogre3D AVIE* renderer

The renderer for *AVIE* that has been developed uses *Ogre3D* as graphics engine. The object oriented nature of the *Ogre3D* engine made the modular implementation of the renderer possible for *AVIE*, which simplifies programming any application. *Ogre3D* is powerful and open source which allows total control through the entire rendering pipeline that runs graphic applications efficiently thanks to its internal management of the scene graph. It also makes portability easy, since an application in *Ogre3D* only needs the executable file and the dynamic libraries employed to run without problems. By using such an engine a lot of possibilities for future improvements can be explored since it allows the incorporation of plug-ins for many different purposes (described later in this chapter).

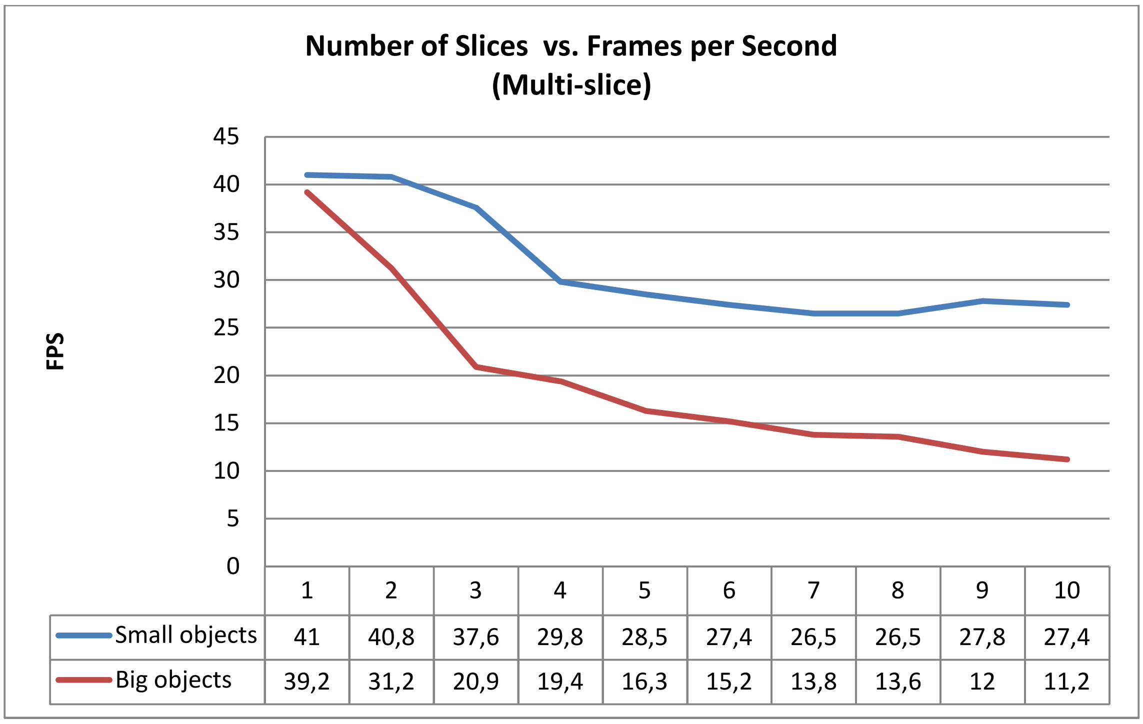
Two types of projections have been implemented as independent camera classes, which abstract the programmer from viewport, view volumes, and many other aspects that become complicated in cylindrical projections. These two cameras are based in a Multi-Slice and cylindrical shader based approach.

5.2.1 Multi-Slice renderer for cylindrical panoramic projections

The multi-slice camera implemented for this system allows the user to choose how many planar perspective projections to use for the task, obtained as output two textures (for left and right eye) with final images composed by the different outputs of the single cameras. If enough slices are used, discontinuities are imperceptible but using too many slices normally has a great impact in terms of performance. Consequently, there is a trade-off between computational efficiency and image quality.

In order to see how the frame rate is affected by using more or less slices a benchmark scene has been prepared for that purpose. This scene displays 24,000 textured triangles all being contained within the displayable view volume (65º horizontally) while benchmarking. The scene in rendered for both eyes, so that it raises the count to calculating 48,000 textured triangles for every single frame. The computer used for this test features a 64 bit *Intel(R) Core(TM) i7* processor at 2.67 Ghz, with 8 GB of RAM with a *Nvidia(R) 330M* graphics card with 512 MB of RAM.

The results obtained are shown in the plot (fig. 83):



	1	2	3	4	5	6	7	8	9	10
Small objects	41	40,8	37,6	29,8	28,5	27,4	26,5	26,5	27,8	27,4
Big objects	39,2	31,2	20,9	19,4	16,3	15,2	13,8	13,6	12	11,2

Fig 83: Frame rate for a scene rendered using different number of slices.

The analysis of the data plotted shows that the tendency of the frame rate depends on the number of rendering passes, which is determined by the number of slices. However, two different tests were conducted with smaller and larger triangles. The reason for this is that if the objects are very small, it is easy to leave them out of the rendering process (view frustum culling) for most of the thin slices. On the contrary, when large triangles are displayed they intersect with many more view volumes of the different slices and consequently those vertices get computed for many slices, which implies repeating calculations. This explains why with large polygons the frame rate becomes much more affected when more slices are included; it also explains why with a single slice the frame rates are very similar (in the case of a single slice the same number of vertices are transformed in only one rendering pass).

However, not all the configurations displayed in the graphic produce satisfactory results. Using one slice does not make sense because it is the same as using a normal perspective projection. When using 2 or 3 slices the discontinuities are too obvious and the results (also for the omnistereo) are very poor. For this reason 4 slices has been chosen as the minimum number of slices for which a reasonable quality is obtained while keeping a satisfactory performance.

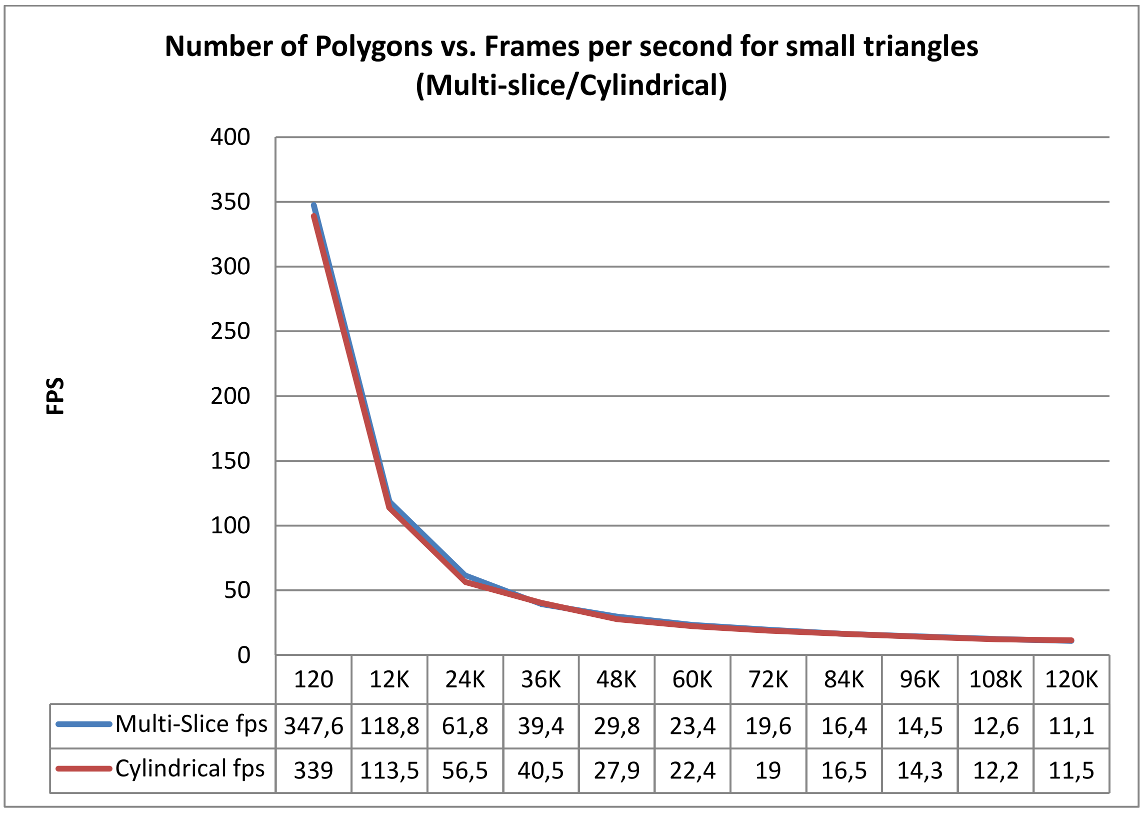
Nevertheless, in the graphic shown, when 4 slices are used the frame rate obtained for small triangles is 29.8 which is good for real time applications, but when large objects are displayed frame rate gets reduced to 19.4. It is important to note that the benchmarking performed tries to illustrate the impact that the slices have over the number of vertex calculations, however many other factors matter. For example, if illumination techniques are being employed every rendering pass becomes more expensive and for that reason the multi-slice approach is too prone to be affected by aspects not only related with the number of polygons, but also with the distribution and size. These problems were identified during the development of the system and for this reason a new kind of camera was implemented (see next section).

5.2.2 Shader-based renderer for cylindrical panoramic projections

The method proposed and implemented in this thesis for calculating real curved projections is considered by the author one of the most important achievements of the overall system. The main motivation for working in this line was sustained by the many problems that have to be always considered when using a multi-slice approach. For example, the risk of obtaining perceptible artifacts (see fig. 63) under some circumstances is always exists. For example, when small objects are very close to the camera, imprecision on the border regions between slices becomes very obvious. It also occurs, that some effects are more difficult to implement or more expensive due to the fact that the discontinuities should not be obvious to the user.

By using a real curvilinear projection (as opposite to approximating it with thin linear perspectives) precision problems disappear, and a seamless image that offers a better omni-directional stereo is achieved at all times, needless to care about special situations like closeness of objects. The problem is that modern graphic cards are highly optimized for planar perspectives, but not for curvilinear ones. In fact they do not offer hardware support for the calculation and everything has to be done within a vertex shader. However, as a move is made towards modern graphic cards, more functionality is moved to the programmable rendering pipeline anyway, so this will be less of an issue in a middle to short term. The implementation of the technique proposed works in real time in current graphic cards, since it is able to convert vertices from the world space to the clip space, which is the output expected from the vertex shader.

A comparison in performance is given below (see fig. 84). The scene used in this case contains a different number of textured polygons, from 60 to 60,000 of them. This implies that the polygons actually being projected to the curved surface are double (since the process is done for both eyes), and hence the number of polygons increases from 120 to 120.000. The computer used for the test is the same as that in the previous benchmark.

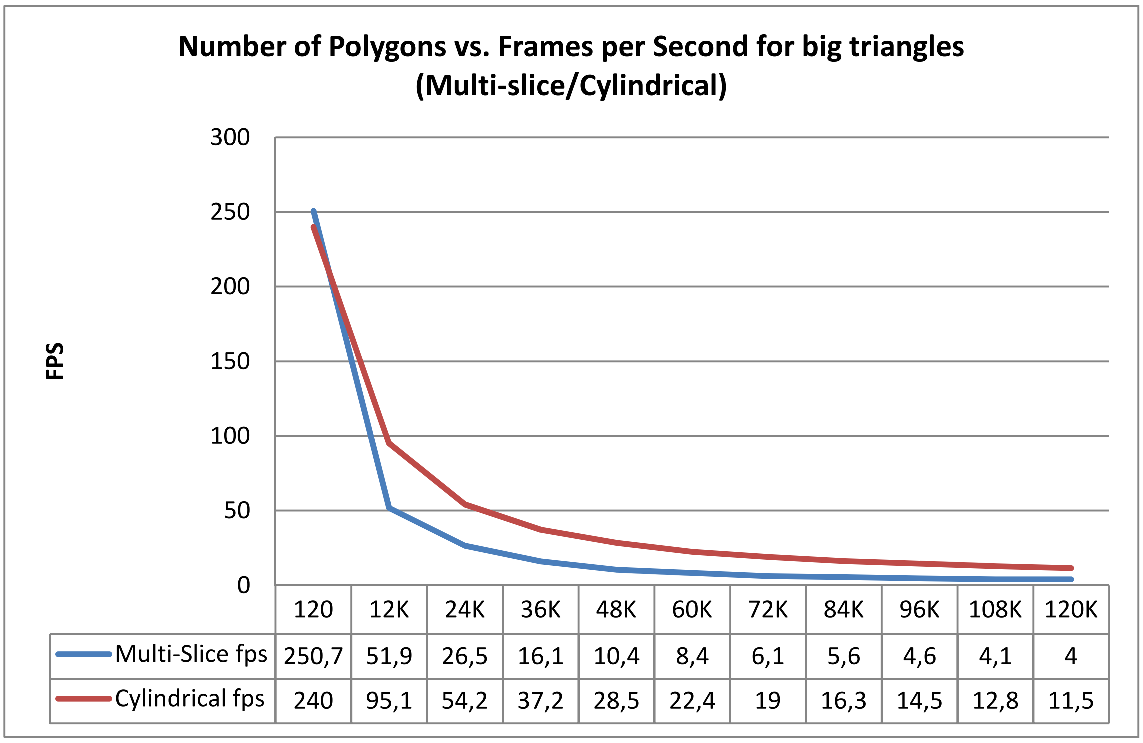


	120	12K	24K	36K	48K	60K	72K	84K	96K	108K	120K
Multi-Slice fps	347,6	118,8	61,8	39,4	29,8	23,4	19,6	16,4	14,5	12,6	11,1
Cylindrical fps	339	113,5	56,5	40,5	27,9	22,4	19	16,5	14,3	12,2	11,5

Fig 84: Frame rate of the cylindrical shader-based approach plotted against the multi-Slice; results with small triangles

As the plot shows, the results are very similar and not very significant differences can be identified. This is already a good result since the performance is practically the same for both multi-slice and shader-based approaches, but shader-based) offers a better image quality and better omnistereo.

However, the results display more evidence that the cylindrical method is favourable if a slightly different test is performed. The previous plot (fig. 84) shows the results obtained when displaying small objects. In the previous section it was illustrated how the larger the objects, the less performance the multi-slice approach offered (because large objects are fully or partially contained in more view frustums and consequently their coordinates are transformed many times in different rendering passes). This situation becomes even worse if more slices are used (and this condition is necessary to obtain better quality in a multi-slice approach) since large objects would be transformed as many times as view frustums contain the object (and obviously the thinner the slices, the more intersections). To show this problem, the same scene, but scaling the objects to make them larger, was used to benchmark the performance of both methods (see fig. 85).



	120	12K	24K	36K	48K	60K	72K	84K	96K	108K	120K
━ Multi-Slice fps	250,7	51,9	26,5	16,1	10,4	8,4	6,1	5,6	4,6	4,1	4
━ Cylindrical fps	240	95,1	54,2	37,2	28,5	22,4	19	16,3	14,5	12,8	11,5

Fig 85: Frame rate of the cylindrical shader-based approach plotted against the multi-Slice. In this case, bigger objects are displayed than those in fig. 84.

This time, the difference in frame rate becomes more obvious. The difference in frame rate with larger or smaller objects is quite reduced; sometimes there is no difference at all (compare results for cylindrical method for both fig. 84 and 85). This makes sense, since the number of vertex calculations that have to be performed in total remains the same. On the contrary, the impact that the size of the objects has in the multi-slice approach´s performance is much more significant, due to the fact that there are many operations that are being performed more than once.

In addition to this, if there are additional calculations to perform per vertex, the situation obviously worsens. For example, if lighting is on, there would be more calculations to do for each vertex, and consequently the repetition of operations that occur in the multi-slice method would become even more expensive, unlike what occurs with the shader-based method (see next section).

5.2.2.1 Additional features implemented for the shader-based cylindrical camera:

When there are calculations to do involving light, discontinuities are even more susceptible to appear with a multi-slice camera. This is exactly what happens when calculating shines. The amount of specular light perceived by the user depends on the observation position. One of the reasons because of discontinuities occur is the difference in the eye positions for two slices that are associated with the same vertex. The cylindrical method avoids the problem because every single vertex is projected just once and onto the curved surface, which is the same for the whole panoramic image.

In order to demonstrate this, two extensions to the shader have been implemented. The first of them is *per-pixel Phong´s illumination*. The adaptation of the technique for a cylindrical projection was very straight-forward and it did not add any additional overload to the necessary calculations for a normal planar projection (see fig. 86).

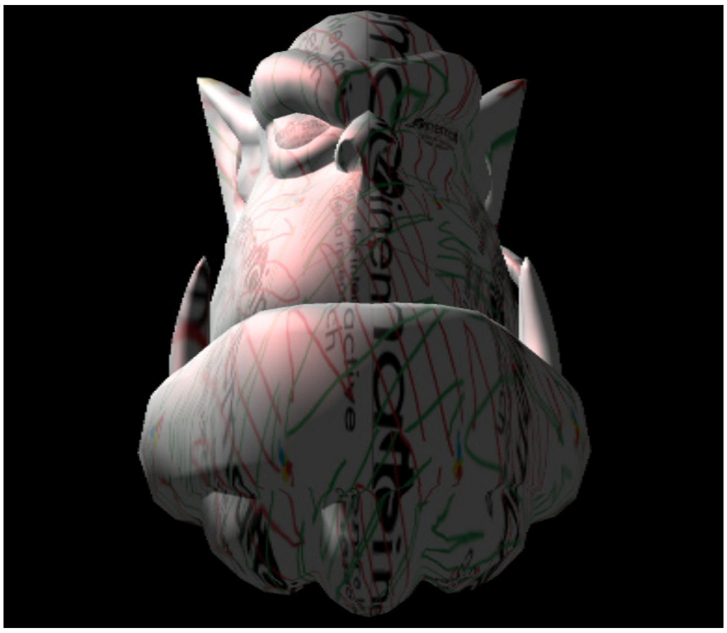


Fig. 86: *Phong* shading example using the curvilinear projection.

*Phong´*s illumination is relatively expensive to compute since it requires additional calculations per pixel and vertex. Hence, *Phong´*s shading technique is even more expensive for a multi-slice camera since some calculations for some vertices are performed more than once. In addition to being more expensive the results are not as accurate as the cylindrical shader.

Another extension implemented is the *Bump-mapping* technique. In the same way it occurred with *Phong´*s shading it does not require more computations than the same technique applied to a regular linear projection.

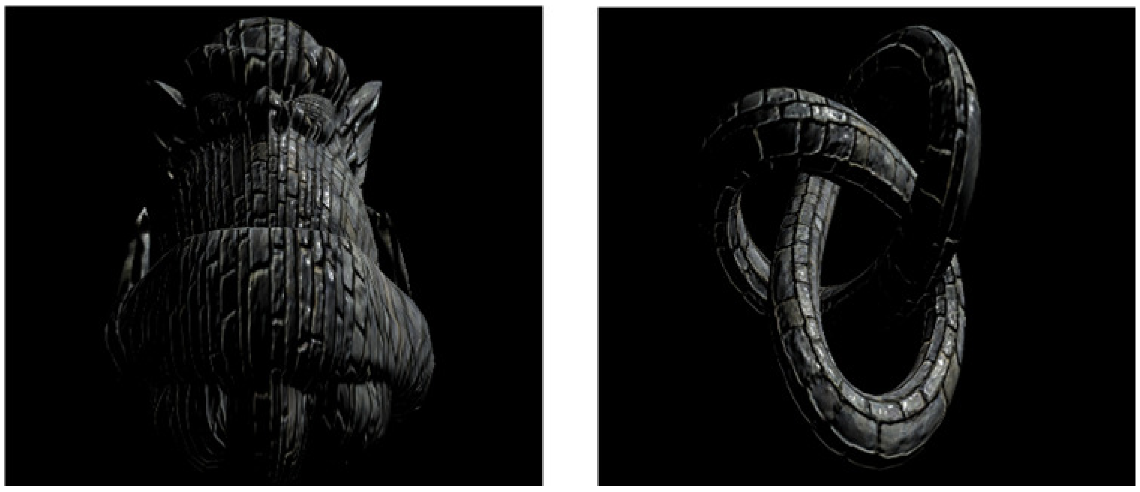


Fig. 87: *Bump-mapping* examples using the curvilinear projection.

Ogre3D offers the possibility of calculating the tangent vectors very efficiently, which are necessary for calculating the illumination using a map of normals. So, the implementation of this method was also very straight-forward.

5.2.3 Other features of the renderer:

The renderer implemented is also prepared for working in clustering mode. This means that, if the option is enabled, the frame buffers are swapped simultaneously with the rest of the computers in the network. The tests of the system in the *AVIE* system have demonstrated the effectiveness of the synchronisation at frame level. The system also allows the server to add/remove objects such as images, video (also synchronised), panoramic images and meshes. Images and video can be 3D, this means that not only the position of the surface that displayed then appears to be in 3D, but there is also a stereographic disparity between the textures displayed for left and right eye.

Other functions implemented help with the process of construction of the scene, since the objects´ positions can be specified in screen coordinates. This implies that the function converts from 2D screen coordinates to the 3D world space coordinates.

One of the advantages of using the Ogre3D engine is the fact that is very easy to include new features due to the multiple *plug-ins* available. Most of these can be adapted easily to work within the render system developed. A few tests have been conducted in order to explore new possibilities. One example is the incorporation of *BSP* scenes [69]. By doing this it is very simple to immerse the user in a complex scene and navigate around, moving the panoramic camera (fig 88).

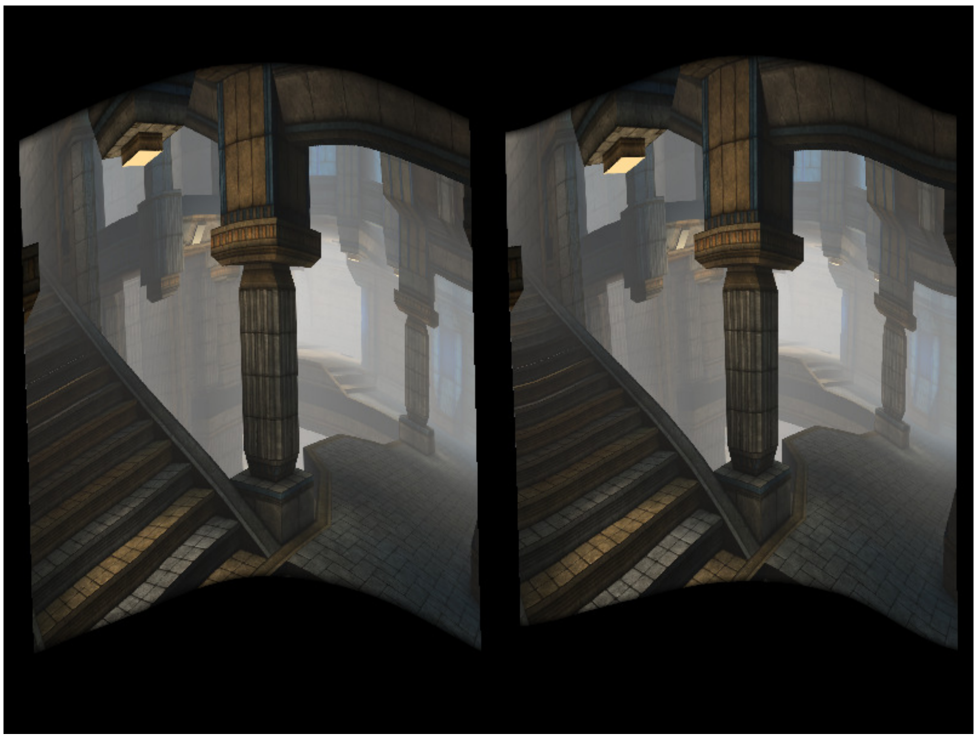


Fig 88: *BSP* scene loaded in Multi-Slice mode with an example warping.

Another plug-in tested that works in the system that could be of interest to work with in order to improve performance, is the *paged geometry* module [70] (see fig 89).

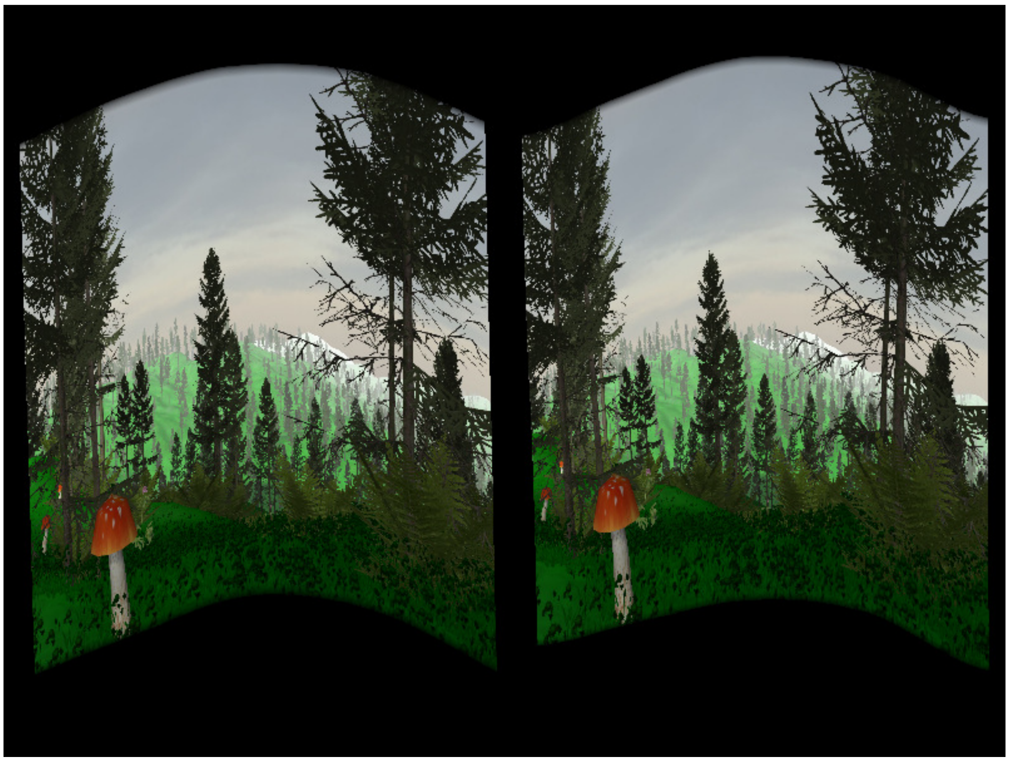


Fig 88: *Paged geometry* scene loaded in Multi-Slice mode with an example warping.

This last plug-in allows to render massive amounts of small meshes, and load them in memory dynamically depending on the entities in the scene that are required. This is particularly useful for outdoor scenes such as forests with millions of trees, bushes, grass, rocks and so on. By incorporating a paged-geometry module into the system, multiple new opportunities would be available for rendering massive outdoor panoramic environments in real time where the user can navigate around within the cylindrical screen.

5.3 The system as a whole

A simple example of use of the whole system is to be described below. All the parts developed take part in this application for *AVIE*; the calibration program for both warping and blending the images of twelve projectors (6 for each eye) projecting a panoramic scene of nearly 360º onto a cylindrical surface. These projectors are connected to a cluster of 6 computers running the renderer program synchronised at frame level. A panoramic object for the background, as well as some other objects will be included in the scene.

The image displayed below (fig. 89) is a screen capture of the initial scene. From the server side, this full-panoramic projection is useful to be able to locate objects around the scene and see where they are located. Initially, a 360º panoramic image is mapped onto a cylinder (object panorama) and placed around the screen. As the screen capture would look as the image itself (since it is a cylinder mapped onto a cylindrical surface projection) some text has also been included in the scene to show the effect of the cylindrical projection.

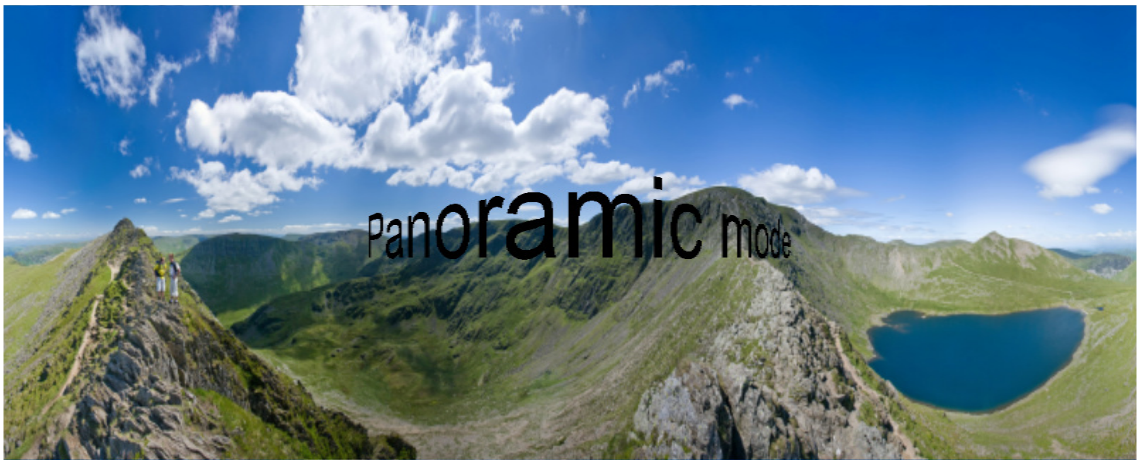
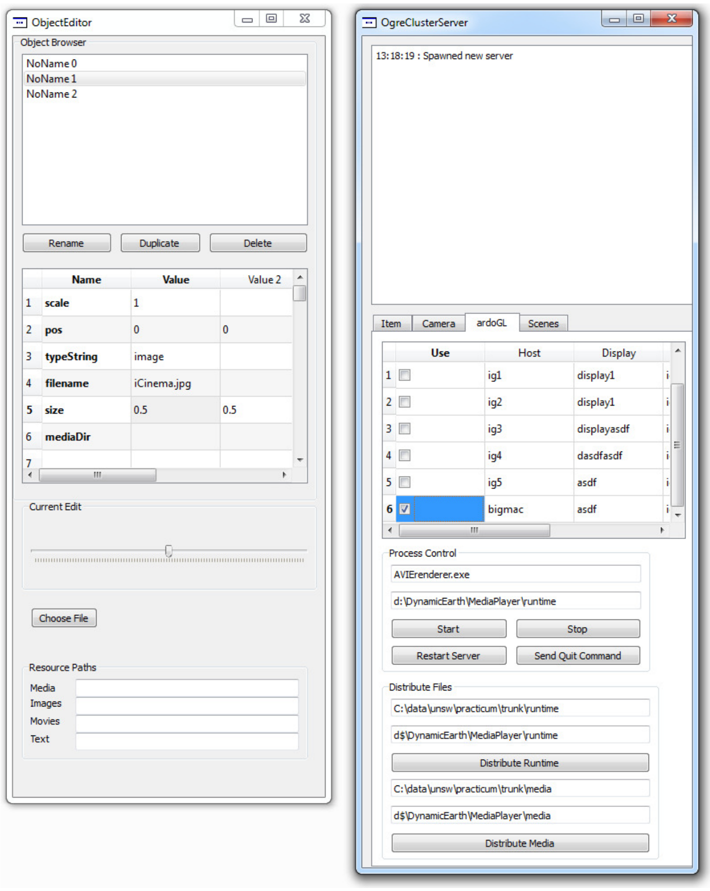


Fig 89: Renderer using panoramic mode in a single computer.

The image, works as a background of the scene, and normally is set at distance parallax zero, since otherwise the background would show a constant disparity for the whole image. Ideally these panoramic images used for background should be 3D as well (different textures for left and right eye). Even if the background is located at parallax distance zero it does not cover other objects, since that object do not write in the depth buffer.

The server application (see fig. 90) can start the renderer program in specific machines in the network. Once this is done, any time that the server adds an image to the scene, the renderer clients will display it as well.

Once the objects are added to the scene, they can be scaled, oriented and located anywhere on the screen by using the object editor (fig. 90a). Scenes can be saved and loaded on other occasions. Normally (but not necessarily) the objects are configured as billboards, in order to make them face the centre of the screen (or any other place). The image below (fig. 91) shows an example of three images added and located around the screen as it can be seen from the server side.



NoName 0
NoName 1
NoName 2

	Name	Value	Value 2
1	scale	1	
2	pos	0	0
3	typeString	image	
4	filename	iCinema.jpg	
5	size	0.5	0.5
6	mediaDir		
7			

```
13:18:19 : Spawned new server
```

	Use	Host	Display
1	☐	ig1	display1
2	☐	ig2	display1
3	☐	ig3	displayasdf
4	☐	ig4	dasdfasdf
5	☐	ig5	asdf
6	☑	bigmac	asdf

Process Control
AVIErenderer.exe
d:\DynamicEarth\MediaPlayer\runtime

Distribute Files
C:\data\unsw\practicum\trunk\runtime
d$\DynamicEarth\MediaPlayer\runtime

C:\data\unsw\practicum\trunk\media
d$\DynamicEarth\MediaPlayer\media

Fig 90: Prototype of server program. Object editor on the left (a) and general controls on the right (b)

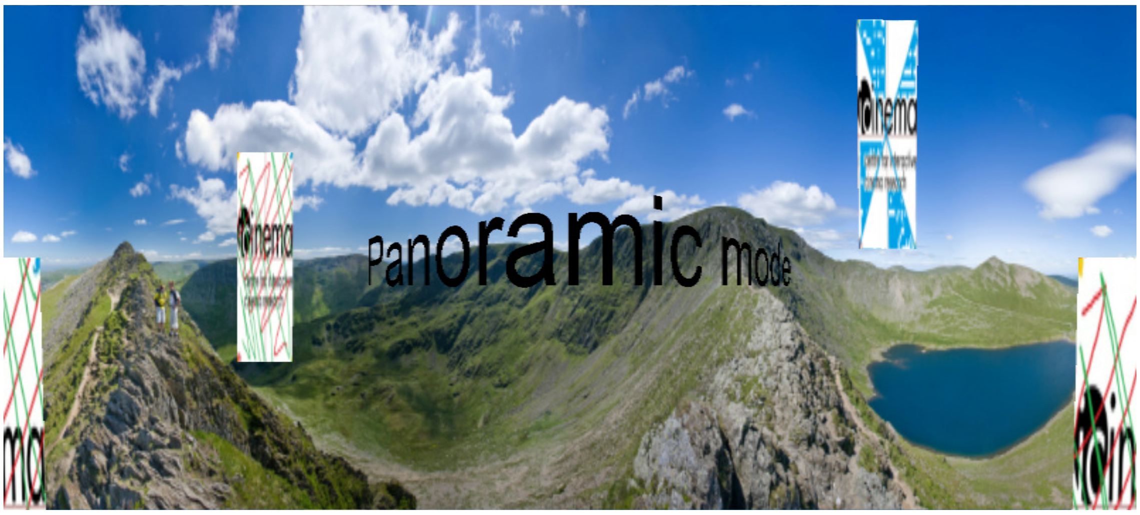


Fig 91: Simple scene seen from the server side with 3 images and a text over a background (on of the images is rendered partially on the left and partially on the right).

The following image (fig. 92) shows the output of the 6 *AVIE* computers for displaying a panoramic scene. After the calibration, the images fit perfectly when projected onto the screen.

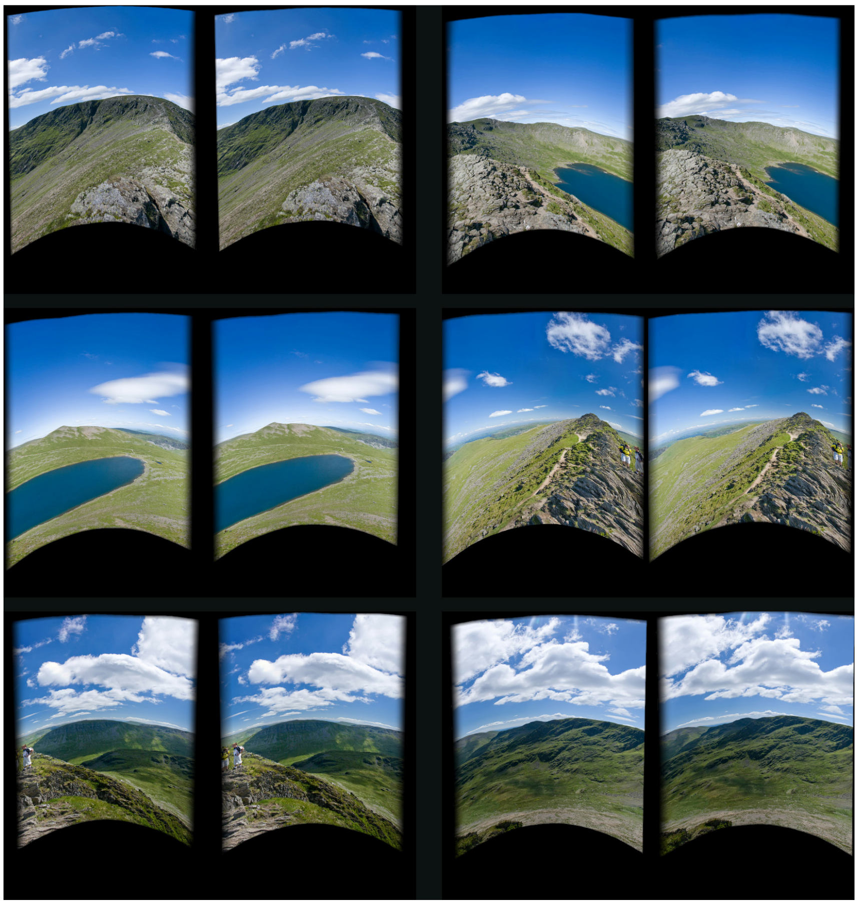


Fig 92: *AVIE* computer´s output images warped and blended.

The quality of the cylindrical projection, the performance obtained, the simplicity of calibration, its high portability and facility to use, determine that the renderer system proposed reaches the goals that were proposed.

Probably, the best proof of this is the fact that this rendering engine is going to be used for a multimedia application in *Melbourne´s museum*, in Victoria (Australia). The project has already started and is under development, but the calibration of the system has been already completed (see fig. 93) in order to start testing. It is programmed that the system will be operative by the end of 2010.

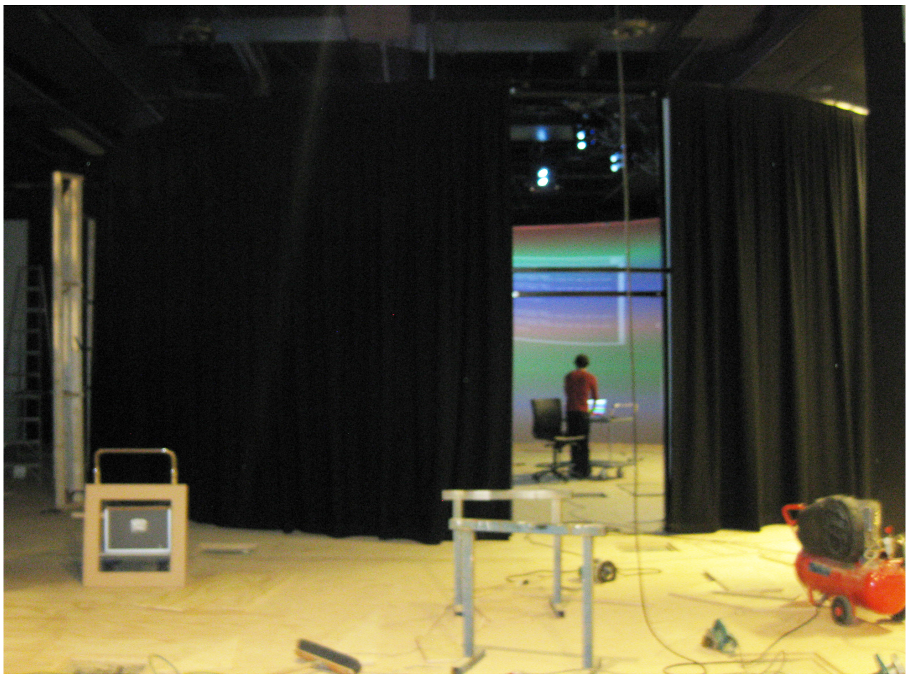


Fig. 93: *AVIE* system in Melbourne´s museum during calibration (site under construction).

5.4 Captures of demonstrations

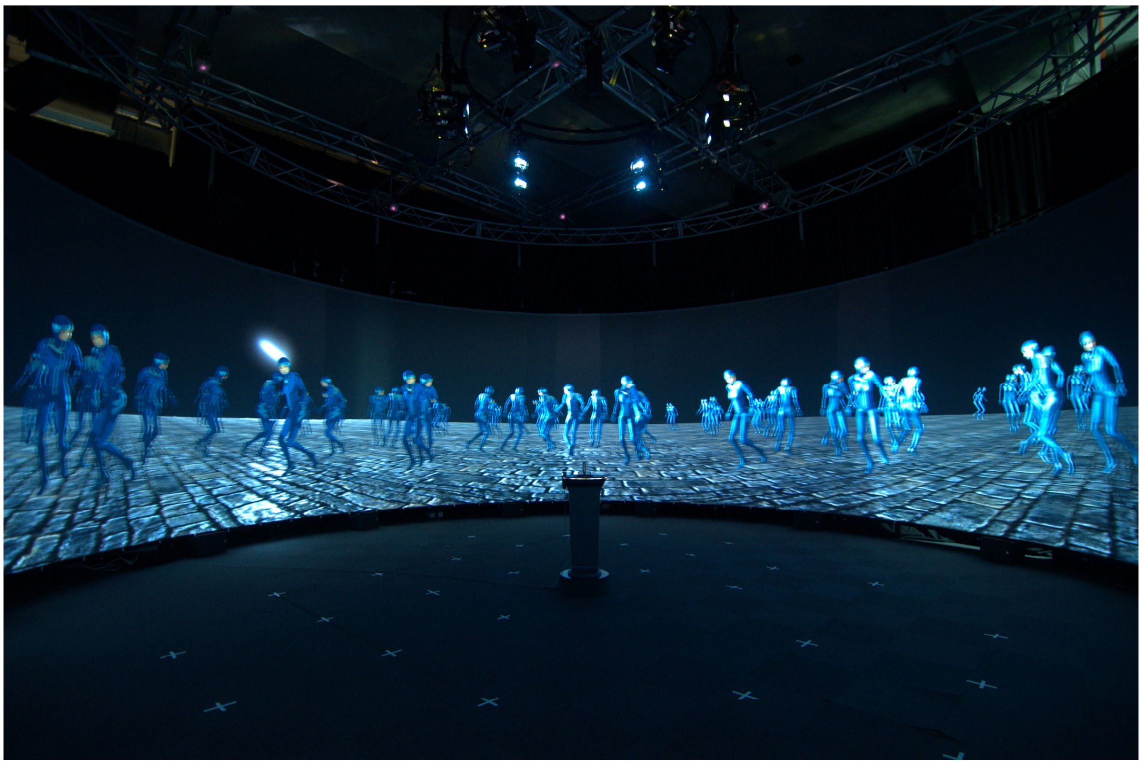


Fig 94: Demonstration in *AVIE* of scene with multiple animations synchronized, omnistereo, *Phong´s* shading and *Bump mapping* (cylindrical shader approach).

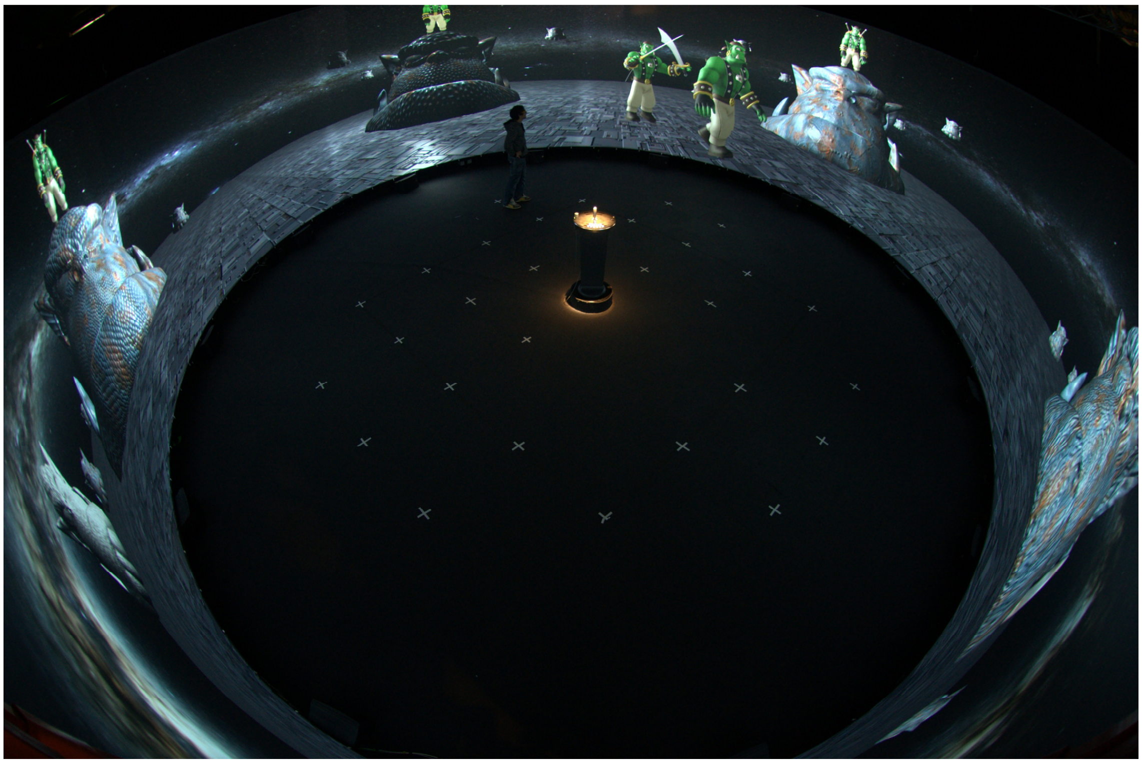


Fig 95: Demonstration in *AVIE* of scene with multiple animations synchronized, omnistereo, *Phong´*s shading and *Bump mapping* (cylindrical shader approach) using *Ogre3D´*s models. Picture taken from above with fish-eye lens.

Chapter 6: Conclusion

During the six months duration of this project, it has been necessary to deal with a very wide range of different problems. The nature of these problems vary greatly and involve aspects such as correctness of the projections, quality of the final image or efficiency. However, two main areas were covered successfully; calibration of the projectors and curvilinear spaces rendering.

The calibration program´s implementation was guided by the analysis of the difficulties that other calibration programs offer as well as their weak points. In the earliest stage of development, it was necessary to investigate in order to find a mathematical model suitable for applying a geometrical correction to the projected images. In fact the first model that was chosen (*Hermite* curves), was finally discarded because it had some problems on the borders of the mesh that could not be solved satisfactorily. *Bezier* curves were also discarded since the type of mesh that offered better control needed to pass through the control points, keep continuity and offer local control. Bezier curves do not fulfil the first requirement and finally *Catmull-Rom* curves (see Analysis chapter) were chosen for the warping. This decision revealed to be appropriate after a while, when it offered the possibility of having better control of the shape of the mesh on the borders.

However, the most problematic part of the calibration system was getting the blending to work. Because of all the difficulties that were being experienced during the tests, to avoid the blending regions being too obvious, extensions were added to the calibration program. For this reason, the final system features 4 control points to define a curved boundary for the blending regions, which facilitates the task since projectors usually are not perpendicular to the projected surface, and consequently the brightness light intensity is not uniform on the surface. After more testing, it was found that the main problem, that caused problems with the blending, was that the projectors that were being used had very different gamma values (even though they were the same model). It was assumed, that the values for the gamma correction (see Analysis chapter) were similar, which is what happens normally. Consequently, a lot of time was dedicated to the blending stage. Later, when calibrating a different *AVIE* system with brand new projectors, the process was much easier (the implementation of the curved blending areas also helped).

The other big issue to deal with was the rendering system. From the early beginning, *Ogre3D* was adopted as the graphic engine to use, since it is open-source, powerful and multi-platform. The first stage consisted in learning how the engine works and the possibilities offered, as well as investigations into how it could be adapted to a system like *AVIE*. After some simple tests and slice rendering prototypes in *OpenGL*, the first renderer in *Ogre3D* was developed. After some investigation and bibliographic revision, it became obvious that the most common approach used for implementing cylindrical projections was to approximate the curved surface with several planar projections, and a multi-slice technique was first implemented. It was already known that this idea produces imprecision in the resulting image, and under some circumstances the slices appear to be very obvious to the observers. It also occured, that in terms of efficiency, the multi-slice approach presented many problems (see chapter Results). At this stage, the idea was to improve the multi-slice technique, and implement some effects avoiding imprecision and other artifacts.

However, while looking for an appropriate solution to the multi-slice approach, more bibliography and papers by other authors were being reviewed in order to find a better way of executing the cylindrical rendering. Some ideas came up, but nothing that was similar enough to what was desired could be encountered. After multiple tests and experimentation, the first consistent real-time cylindrical renderer using a real cylindrical view volume for *AVIE* was implemented. This made it obvious that this approach had multiple advantages (accuracy, better omnistero, efficiency, and so forth) and the development of the system was focused nearly entirely in this new shader-based idea from that moment.

Many other aspects were being taken into consideration as well, such as the synchronisation. A network of computers had to run the application simultaneously and not only synchronised at frame level, but also animations and videos in the scenes should be updated at exactly the same time.

At some point, the development of new features was implemented considering an application for *Melbourne´s Museum* in Victoria (Australia), being this the best testimony of the quality of the system developed. It is also very revolutionary, that a cylindrical renderer for *AVIE* without slices has been implemented for the first time, demonstrating that the goals of the project have been achieved. The system is also able to generate cylindrical omnistereoscopic renderer for panoramic images of 360º with an unprecedented accuracy. Such a system can be employed for many different purposes, such as panoramic video players, scientific visualisation, entertainment, simulation and so forth.

This new renderer system also leaves the door open to many new features and improvements to be developed in the future. For example, the calibration system is very susceptible to being extended to perform the calibration automatically. There is work done by other authors that indicates that this possibility is feasible for the warping method in use. For the cylindrical renderer there is much more to do, since there are multiple shader-based effects that would be of interest to adapt as it has been done with the *Phong shading* and *Bump mapping*. In most cases this can be achieved in a very straight-forward way.

Also, the interaction of the users with the system can be easily improved, by making use of the joystick or orientation sensor of the *AVIE*. There are modules already built for these purposes and their integration would be very straight-forward. The same applies to the tracking systems such as body, head or finger tracking. This information could be used in the renderer proposed in order to enhance interactivity.

Another possibility to investigate is regarding curved projections for single users. In this system omnistereo projections are used, but it has been justified (Analysis chapter) that this is a compromised solution. It would be interesting to track the eyes of the user, rather than the head, in order to be able to calculate a very precise curved projection of the scene. This may be feasible since glasses or HMDs are generally needed for achieving stereo. Probably combining head-tracking with eye-tracking performed by a camera pointing to the eyes would be a possibility.

Also, there is much more to investigate regarding realistic camera projections, rather than only curved projections. It would be interesting to model some physiological characteristics of our vision system in order to enhance the realism and accurate depth perception. In any way, curved projections seem to be the tool to use to achieve more realistic immersive visualisation systems.

Personally, I consider that my time at the *iCinema* research centre has been very professionally enriching. Many of the concepts that I have been dealing with are related with curvilinear projections, which are not yet supported natively by the current graphic hardware. However, I believe that in the next years this is going to change and these kinds of projections will become a standard, since they represent a better way of representing information, and is closer to the way we perceive the world. Developing a visualisation system for a platform like *AVIE* involves dealing with many problems that are common to many visualisation systems. I am now familiar with a variety of these, which include: computer synchronisation, blending, warping, curvilinear projections, omnistereo, and so forth. These concepts are present in many computer graphic applications. I also strongly believe that these concepts are soon to become the target of industry (entertainment, simulation, etc) when looking for more realistic experiences, as has already occurred with 3D technologies. For this reason, the chance that I have had to develop a rendering system from scratch for *AVIE* has contributed many benefits to my professional future.

Appendix 1

Calibration program user's manual

The calibration program is an application able to warp images to adapt them to a curved or irregular projection surface. It also offers the possibility of blending the borders of the image, which is necessary when using several projectors in adjacent areas where the image is partially overlapped. For more information about the implementation of the program, go to the Analysis and Design chapters of this report. The aim of this section is to describe the interface of the program and discuss an example of use.

Commands list:

q/Q	Adds/removes horizontal control points (only warping mode).
w/W	Adds/removes vertical control points (only warping mode).
o/O	Adds/removes horizontal quads to the texture to be warped (only warping mode).
p/P	Adds/removes vertical quads to the texture to be warped (only warping mode).

e/E	Increases/decreases the definition of the mesh (only warping mode).
r/R	Increases/decreases the tension of the curves of the mesh (only warping mode).
v/V	Switches verbose mode, more or less information is displayed.
b/B	Switches between the warping and the blending option (different options for blending can be selected by pressing b repeatedly).
t/T	Increases/decreases the exponent of the blending functions. It only affects an area if it is selected, highlighted in red (only blending mode).
y/Y	Increases/decreases the gamma value (only blending mode).
i/I	Increases/decreases the exponent 'a' value of the blending functions which controls the brightness in the central point of the blending area (only blending mode).
c/C	Activates/deactivates the collapse mode for the borders. It affects the shape of the mesh. No collapse is better for aligning the projection with another ones. If it is only one projection, then collapse mode produces a softer appearance on the borders of the mesh (only warping mode).
m/M	Locks/unlocks the blending control points to move them individually or not. It only affects an area if it is selected, highlighted in red (only blending mode).
s/S	Saves the mesh.
l/L	Loads the previously stored mesh.
j/J	Resets the mesh.
z/Z	Increases/decreases the number of quads overlapping on the left (only warping mode).
x/X	Increases/decreases the number of quads overlapping on the right (only warping mode).

Esc	Exits the program.

Calibration example:

An example of calibration for *AVIE* is will be discussed for future reference. The calibration process is very similar for any other system.

First of all, it is necessary to have a reference in real world units of the screen´s geometry. The easiest way to do this is by placing markers around the projectable surface every X centimetres. Experimentally it was found very useful to use a laser light to aid with the process. Once this is done the projectors are switched on to view which regions are affected. For example, if the projectors of a determinate computer illuminate 20 of the markers located on the screen, then the calibration program has to be configured to show as many quads as markers (keys o/O and p/P). It is important to have a well defined overlap between adjacent projections (for example, two quads). The calibration program is started in every computer in the mode for left and right calibration and with the correct resolution. A script to start all the programs simultaneously is useful for this purpose (it is easier to switch off the right projectors or to swap the screen to black with the key 'b' to see only one mesh at a time).

One by one, the user calibrates the warping. In *AVIE* it has been observed that 6 by 4 control points is quite a good proportion and offers a good control while keeping the calibration simple. The mode with no collapse of the points on the borders is also easier to configure. Other parameters such as tension, and definition of the mesh, can be tuned as well. Overlapping on the left and right are important to set or the latter stage will not work properly.

Once the warping is completed for the left eye, the right eye is very quick since it only consists in dragging the control points over the ones for the left eye, and the mesh should fit perfectly.

It is also a good idea to save the meshes every once in a while to make sure that no information is lost.

At this point, the geometry is well aligned but the intensity of light is much higher in the overlapping areas. Blending is the process to fix this problem. When the user switches to blending mode, it is probably a good idea to start the calibration with the colours texture since it offers a good representation of the chromatic range. Otherwise the white texture or a texture decided by the user can be chosen (with the key ‘b’). Parameters to take into account for the blending together with the control points, are the gamma and the exponent of the function (as the most important ones) but also the ‘a’ value to control the brightness in the central area of the blending. Probably it is a good idea to start moving the 4 that define the blending curves on the sides of the image all together and once the result is good, improve it moving the points one by one (switching between the modes with the key ‘m’).

Once this is done, it is necessary to check the configuration files created. If the size of the squares is not the correct one, then it has to be set manually, so too do the total number of quads in the screen.

After the individual calibration for every computer is completed (warping and blending), meshes need to be saved. After this the script that calculates the offset angles needs to be run from the server side. This script overwrites the configuration files with the new values and also distributes the files copying them to the runtime folders in which the executable programs are.

Appendix 2

Cylindrical renderer for *AVIE*, programmer´s manual

It is the purpose of this manual to give a brief introduction to the *Ogre3D AVIE* renderer implemented. Needless of complicate changes, the renderer source code allows the quick implementation of new scenes, ready to be displayed in a cylindrical environment.

If the calibration files area already available (see Appendix 1) and located in the same folder as the renderer executable file (together with all the necessary *dll´*s, configuration files) and the media folder is accessible, then the program should already be able to run.

First of all, the running mode has to be set. If the *define* label '*USE_CLUSTERING*' in '*AVIErenderer.h*' is not commented, then the program runs in synchronised mode, and expects to be started from a server computer who will provide its name through command line. The main difference of this mode is that the image buffer swaps only when the rest of the computers of the network are ready. Otherwise, if the label is commented, the application will run in standalone mode.

Secondly, the projection camera has to be set. Also in '*AVIErenderer.h*' there is another label for that purpose. If '*USE_CYLINDRICAL_RENDERING*' is not commented, a real cylindrical projection is used for this computer. This is the recommended option, since it offers better quality and performance, but it is up to the programmer to decide using a multi-planar projection based camera by leaving the label commented.

There are some methods implemented in the class '*AVIErenderer*' that include objects to the scene. These can be used, but it does not make much sense since they are specifically though to be called by the server application.

If the implementation of a new scene is to be done locally, then the code should be included in the method '*createScene*'. An *Ogre3D* programming manual can be consulted to know how to add and place objects in the scene (Junker, 2006 [63]). Objects can also be placed by using the '*get3DCoordsFromScreenCoords'* function accessible from the '*AVIErenderer'* class, that converts a 2D point in screen coordinates (absolute or normalised) to a 3D scene point.

It is important to note that if the cylindrical rendering is being used, the material has to be set in a different way. Probably, the most straight forward way is by adding '*OgreEntityAVIE'* objects since they implement a method that aids with the material setting. Different materials are provided, so the programmer can use them or modify them to include new features. '*CylindricalTextureMapping'* (for normal texture mapping), '*CylindricalTextureBumpMapped'* (for bump mapping, the normal texture has to be provided as well) and '*CylindricalTexturePhongShading'* (for per-pixel illumination) are the names of the materials that can be selected together with a texture.

Most of the features of *Ogre3D* are available and work fine for the *AVIE* renderer. An exception is the Ogre´s billboard objects implementation. If an object wants to act as a *billboard* or follow the camera it has to be added to the camera directly by using the methods '*addBillboard'* or '*addFollower'*, and set the update of these elements active for the camera with '*activeBillboardUpdate'* and '*activeFollowerUpdate'*. These objects can be also unattached by using the corresponding methods.

Appendix 3

Cylindrical and spherical-polar coordinates

Curvilinear coordinates [71] are coordinate systems for Euclidean space in which the coordinate lines may be curved. These coordinates can be derived from a set of Cartesian coordinates by using a transformation that is locally invertible at each point. Consequently, a given point in the Cartesian system can be converted to its curvilinear coordinates and back. In the same way a Cartesian coordinate surface is a plane (i.e. $z=0$ defines the x-y plane), in curvilinear coordinates a surface is obviously a curved surface (i.e. $r=1$ in spherical coordinates is the surface of a unit sphere). Two well-know examples of curvilinear coordinate systems are cylindrical and spherical-polar coordinates for $\boldsymbol{R}^3$.

Cylindrical coordinate system:

A cylindrical coordinate system [72] is a three-dimensional coordinate system that specifies point positions by the distance from a chosen reference axis (ρ), the direction from the axis relative to a chosen reference direction (φ) and the distance from a chosen reference plane perpendicular to the axis (Z). (See fig. 96).

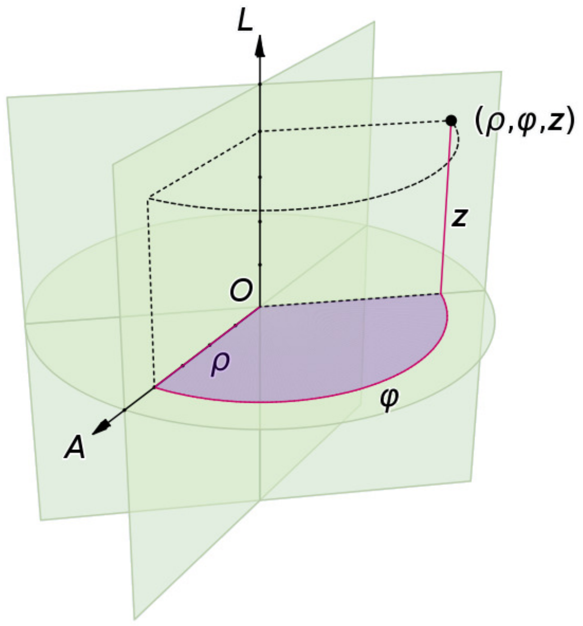


Fig 96: Cylindrical coordinates (from *Wikipedia*).

For the conversion between cylindrical and Cartesian coordinate systems, it is useful to assume that the reference plane is the Cartesian *x-y* plane (*z=0*), and the cylindrical axis is the Cartesian *z* axis. Then the *z* coordinate is the same in both systems, and the correspondence between cylindrical (ρ, φ) and Cartesian (x, y) are the same as for polar coordinates. Then the following relations can be obtained:

$$x = \rho \cos \varphi$$
$$y = \rho \sin \varphi$$
$$z = z$$

in one direction, and

$$\rho = \sqrt{x^2 + y^2}$$

$$\varphi = \begin{cases} 0 & \text{if } x=0 \text{ and } y=0 \\ \arcsin(\frac{y}{\rho}) & \text{if } x \geq 0 \\ -\arcsin(\frac{y}{\rho}) + \pi & x < 0 \end{cases}$$

$$z = z$$

in the other. Assuming angles $[-90°,+90°]$ for the *arcsin* function.

Spherical coordinate system:

A spherical coordinate system [73] is a three-dimensional coordinate system where the position of a point is specified by three values; the radial distance of that point from a fixed origin (*r*), its inclination angle measured from a fixed zenith direction (*θ*), and the azimuth angle of its orthogonal projection on a reference plane that passes through the origin and is orthogonal to the zenith (*φ*), measured from a fixed reference direction on that plane. (See fig. 97).

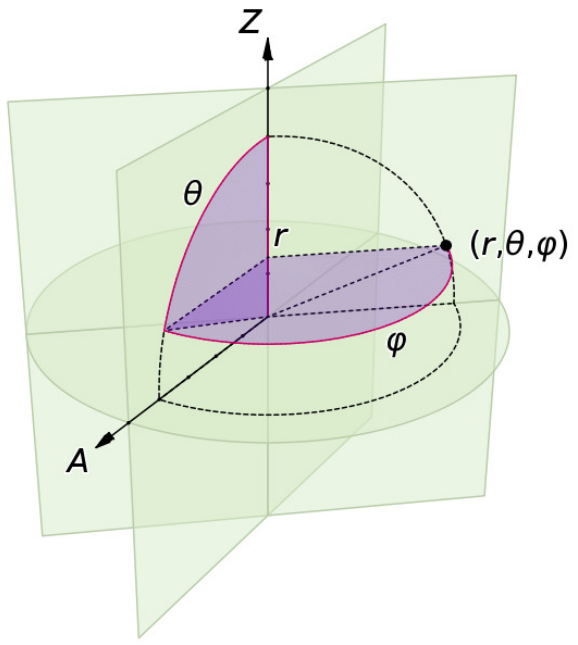


Fig 97: Spherical coordinates (from *Wikipedia*).

For conversion between Cartesian coordinates (*x,y,z*) and spherical coordinates (*r,θ, φ*) the following formulas can be used:

$$x = \sin\theta \, \cos\varphi$$
$$y = r \sin\theta \, \sin\varphi$$
$$z = r \cos\theta$$

in one direction, and

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\theta = arcos \frac{z}{\sqrt{x^2 + y^2 + z^2}}$$

$$\varphi = atan2(y\,, z)$$

in the other. The *atan2* function is a variant of the *arctan* function that returns the angle from the x-axis to the vector (x, y) in the full range (-π, π]. One cannot use $arctan(y\,, z)$ to obtain φ, because it returns the same angle for (x, y) and $(-x, -y)$.

$$atan2(y, x) = 2\arctan \frac{y}{\sqrt{x^2 + y^2} + x}$$

or alternatively

$$atan2(y,x) = \begin{cases} arctan(\frac{y}{x}) & x > 0 \\ \pi + arctan(\frac{y}{x}) & y \geq 0, x < 0 \\ -\pi + arctan(\frac{y}{x}) & y < 0, x < 0 \\ \frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ undefined & y = 0, x = 0 \end{cases}$$

Bibliography

[1] Milgram, Paul; H. Takemura, A. Utsumi, F. Kishino (1994). “Augmented Reality: A class of displays on the reality-virtuality continuum”. Proc. SPIE Vol. 2351, p. 282-292, Telemanipulator and Telepresence Technologies, Hari Das; Ed.

[2] Holliman, Nick (2005). “3D display systems”.

[3] Halle, Michael (1997). “Autostereoscopic displays and computer graphics”. Computer Graphics, ACM SIGGRAPH, 31(2), May 1997, pp. 58-62.

[4] Benton, Stephen (1982). “A survey of holographic stereograms”. Proceedings Processing and Display of Three-Dimensional Data, 367, SPIE, pp. 15-19, 1982.

[5] Graham, Saxby (2003). “Practical Holography”, Third Edition. Taylor & Francis
ISBN: 978-0-7503-0912-7, eBook ISBN: 978-1-4200-3366-3.

[6] Blundell, B. & Schwarz, A. (2000). “Volumetric Three-Dimensional Display Systems”, John Wiley & Sons. ISBN 0-471-23928-3.

[7] Payatagool, Chris (2007). “Three Dimensional Images in the Air: Visualization of Real 3D images using Laser Plasma”. The National Institute of Advanced Industrial Science and Technology (Japan).

[8] Jones, Andrew; McDowall, Ian; Yamada, Hideshi; Bolas, Mark; Debevec, Paul (2007). “Rendering for an Interactive 360º Light Field Display”. SIGGRAPH 2007.

[9] Helmut, Jorke; Fritz, Markus (2005). “*Infitec*: A new stereoscopic visualization tool by wavelength multiplexing imaging”. Journal of Three Dimensional Images L0436A. ISSN:1342-2189. Vol.19; No.3; Pages.50-56.

[10] Helmut, Jorke; Fritz, Markus (2006). “Stereo projection using interference filters”. Proc. SPIE Int. Soc. Opt. Eng. 6055, 6055G

[11] Turnet, T; Hellbaum, R (1986). “LC shutter glasses provide 3-D display for simulated flight Source” Information Display archive. Volume 2 , Issue 9

[12] Raskar, R., Beardsley, P.A., "A Self-Correcting Projector", IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), December 2001.

[13] R. Sukthankar, R. Stockton, and M. Mullin., “Smarter presentations: Exploiting homography in camera-projector systems,” In Proceedings of International Conference on Computer Vision, 2001.

[14] Eric W. Weisstein. "Homography." From MathWorld—A Wolfram Web Resource. http://mathworld.wolfram.com/Homography.html.

[15] Lee, Cha; Bostandjiev, Svetlin (2009). “Automated calibration Method for Multi-Projector Curved Displays”.

[16] Van Baar, Jeroen; Willwacher, Thomas; Rao, Srinivas; Raskar, Ramesh (2003). “Seamless multi-projector display on curved screens”. EGVE, Vol 39. Proceedings of the workshop on Virtual environments. Pages: 281 - 286 .

[17] Sun, Wei; Sobel, Irwin; Culbertson, Bruce; Gelb, Dan; Robinson, Ian (2008). “Calibrating multi-projector cylindrically curved displays for ‘wallpaper’ projection”. International Conference on Computer Graphics and Interactive Techniques.

[18] Johnny C. Lee , Paul H. Dietz , Dan Maynes-Aminzade , Ramesh Raskar , Scott E. Hudson (2004). Automatic projector calibration with embedded light sensors. Symposium on User Interface Software and Technology archive Proceedings of the 17th annual ACM symposium on User interface software and technology. Pages: 123 – 126.

[19] Cruz-Neira C., Sandin D.J., DeFanti T.A., Kenyon R.V. and Hart J.C., "The CAVE: Audio Visual Experience Automatic Virtual Environment" Communications of the ACM, 35 (6): 65-72, 1992.

[20] Cruz-Neira C., Sandin D.J. and DeFanti T.A., "Surround-Screen projection Based Virtual Reality: The Design and Implementation of the CAVE", Computer Graphics, 27: 135-142, 1993.

[21] Kolb, Andreas; Lambers, Martin; Todt, Severin; Cuntz, Nicolas; Rekz-Salama, Christof (2009). “Immersive rear projection on curved screens”, VR2009: 285-286.

[22] Kuchelmeister, Volker; Shaw, Jeffrey; McGinity, Matthew; Del Favero, Dennis; Hardjono, Ardrian (2009) “Immersive Mixed Media Augmented Reality Applications and Technology”. Advances in multimedia information processing, PCM 2009.

[23] Bourke, Paul (2005). “Using a spherical mirror for projection into immersive environments (Mirrordome)”. Graphite (ACM Siggraph). Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia. pp 281-284

[24] Bourke, Paul (2007). Isphere http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/domefisheye/isphere/

[25] Bourke, Paul (2002). “Cube2dome. Creates fisheye images from cubic maps”. http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/domefisheye/cube2dome/

[26] Bourke, Paul (2010). “Convert spherical projections to cylindrical projection”. http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/sph2pan/

[27] Liu, Zicheng; Cohn, Michael (2002). “Real-Time Warps for Improved Wide-Angle Viewing”. Microsoft Research, MSR-TR-2002-110.

[28] McGinity, Matthew; Shaw, Jeffrey; Kuchelmeister, Volker; Hardjono, Ardrian, Del Favero, Dennis (2007). “*AVIE*: a versatile multi-user stereo 360º interactive VR theatre”. ACM International Conference Proceeding Series; Vol. 252. Article no.:2.

[29] "Genlock" From wikipedia. http://en.wikipedia.org/wiki/Genlock

[30] Penny, S; Smith, J; Bernhardt, A. (1999). “Wireless full body tracking in the cave”. ICAT 99 Conferece Proceedings.

[31] Barker, T. (May 2007). "Adapting a Model of Duration: The Multitemporality of T_Visionarium II" M/C Journal, 10(2). <http://journal.media-culture.org.au/0705/14-barker.php>.

[32] Bennett, J. (2007). “T_Visionarium: A User's Guide”. Sydney/Karlsruhe: UNSW Press/ZKM.

[33] Del Favero, D., Brown, N., Shaw, J. & Weibel, P. (2005b). "T_Visionarium: The Aesthetic Transcription of Televisual Databases" in U. Frohne & M. Schieren (Eds). Present, Continuous, Past (pp. 132-141). Springer: New York.

[34] "Keystone effect." From wikipedia. http://en.wikipedia.org/wiki/Keystone_effect

[35] Bourke, Paul (2004) "Image Warping for projection onto a cylinder" http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/stereographics/cylinder/

[36] Salomon, David (2006). “Curves and Surfaces for Computer Graphics”. Springer, 460 pp. ISBN: 9780387241968

[37] Bourke, Paul (2004). “Edge blending using commodity projectors”. http://local.wasp.uwa.edu.au/~pbourke/texture_colour/edgeblend/

[38] "Gamma correction" From wikipedia. http://en.wikipedia.org/wiki/Gamma_correction

[39] Lingard, Brian (1995) “A tutorial on Time-Multiplexed Stereoscopic Computer Graphics”. http://web.cs.wpi.edu/~matt/courses/cs563/talks/stereohtml/stereo.html

[40] Foley, Van Dam, Feiner, Heghes. Computer Graphics: principles and practice. Second Edition in C. International Edition. Pearson, Addison Wesley. 1997.

[41] "Anamorphosis" From wikipedia. http://en.wikipedia.org/wiki/Anamorphosis

[42] Collier, J.M. (1975) “Linear Perspective in Flemish paintings and the art of Petrus Christus and Dirk Bouts”.

[43] Estes, John E; Hemphill, Jeff (1998) “Camera systems”. Remote sensing core curriculum. Volume 1 – module 3.1. http://rscc.umn.edu/rscc/v1m3a.html

[44] Glaeser, Georg (1999) “Extreme and subjective perspectives”. The Visual Computer,. Springer.

[45] "Depth perception" From wikipedia. http://en.wikipedia.org/wiki/Depth_perception

[46] Boyd, Danah (2000) “Depth Cues in Virtual Reality and Real World”. The Visual Computer,. Springer.

[47] Greenstone, Brian (2004). “Ultimate Game Programming Guide for Mac OS X”. Pangea Software Inc. ISBN: 0-9761505-0-6.

[48] Bayarri, Salvador (1995) “Computing non-planar perspectives in real time”. Computers & Graphics. Volume 19 , Issue 3. Pages: 431 – 440.

[49] Yang, Yonggao; Cehn, Jim X.; Beheshti, Mohsen (2005). “Nonlinear Perspective Projections and Magic Lenses: 3D View Deformation”. IEEE Computer Graphics and Applications. Volume 25 , Issue 1. Pages: 76 – 84.

[50] "Cube mapping." From wikipedia. http://en.wikipedia.org/wiki/Cube_mapping

[51] Bourke, Paul; Felinto, D. Q. (2010). “Blender And Immersive Gaming In A Hemispherical Dome”. Proceedings of the Computer Games & Allied Technology 10 (CGAT10).

[52] Bourke, Paul (2001). “Computer Generated Angular Fisheye Projections” http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/domefisheye/fisheye/

[53] Trapp, Matthias; Döllner, Jürgen (2008). “A Generalization Approach for 3D Viewing Deformations of Single-Center Projections”. 3rd GRAPP International Conference on Computer Graphics Theory and Applications.

[54] Lorenz, Haik; Döllner, Jürgen (2009). “Real-time Piecewise Perspective Projections”. GRAPP 2009 – International Conference on Computer Graphics Theory and Applications. Pages 147-155

[55] Simon, Andreas; Smith, Randall C.; Pawlicki, Richard R. (2004). “Omnistereo for Panoramic Virtual Environment Display Systems”. Proceedings of the IEEE Virtual Reality 2004. Page 67.

[56] Segal, Mark; Akeley,Kurt (2010) “The OpenGL Graphics System: A Specification (Version 4.1)”

[57] Bourke, Paul (2006) “Synthetic Stereoscopic Panoramic Images”. Visual Systems and Multimedia 2006. Pages 147-155

[58] Salamon, David (2005). “Transformations and Projections in Computer Graphics”. Springer London, pp 145-220. DOI: 10.1007/978-1-84628-620-9.

[59] Bourke, Paul (2009). “Omni-directional Stereoscopic Fisheye Images For Immersive Hemispherical Dome Environments”. Computer Games & Allied Technology 09 (CGAT09), pp 136-143

[60] Glew website. http://glew.sourceforge.net/

[61] Riccio, Christophe (2010) “GLM: Manual” http://glm.g-truc.net/

[62] Bourke, Paul (2002). “Image Warping/Distorsion” http://local.wasp.uwa.edu.au/ ~pbourke/miscellaneous/imagewarp/

[63] Junker, Gregory (2006).“Pro Ogre3D Programming”. Apress ISBN-13: 978-1-59059-710-1

[64] The Ogre Manual v1.7 (‘Cthungha’). http://www.ogre3d.org/docs/manual/

[65] Bink video webpage. http://www.radgametools.com/bnkmain.htm

[66] Davis, Tom (2001) “Homogeneous Coordinates and Computer Graphics”

[67] Kilgard, Mark J. (2003) “The CG tutorial”. Addison Wesley. NVIDIA Corporation

[68] "Texture mapping" From wikipedia.http://en.wikipedia.org/wiki/Texture_mapping

[69] "Bynary Space Partitioning" From wikipedia. http://en.wikipedia.org/wiki/Binary_space_partitioning

[70] Paged Geometry plug-in webpage. http://www.ogre3d.org/tikiwiki/tiki-index.php?page=PagedGeometry+Engine

[71] "Curvilinear coordinates" From wikipedia. http://en.wikipedia.org/wiki/Curvilinear_coordinates

[72] "Cylindrical coordinate system" From wikipedia. http://en.wikipedia.org/wiki/Cylindrical_coordinate_system

[73] "Spherical coordinate system" From wikipedia. http://en.wikipedia.org/wiki/Spherical_coordinates