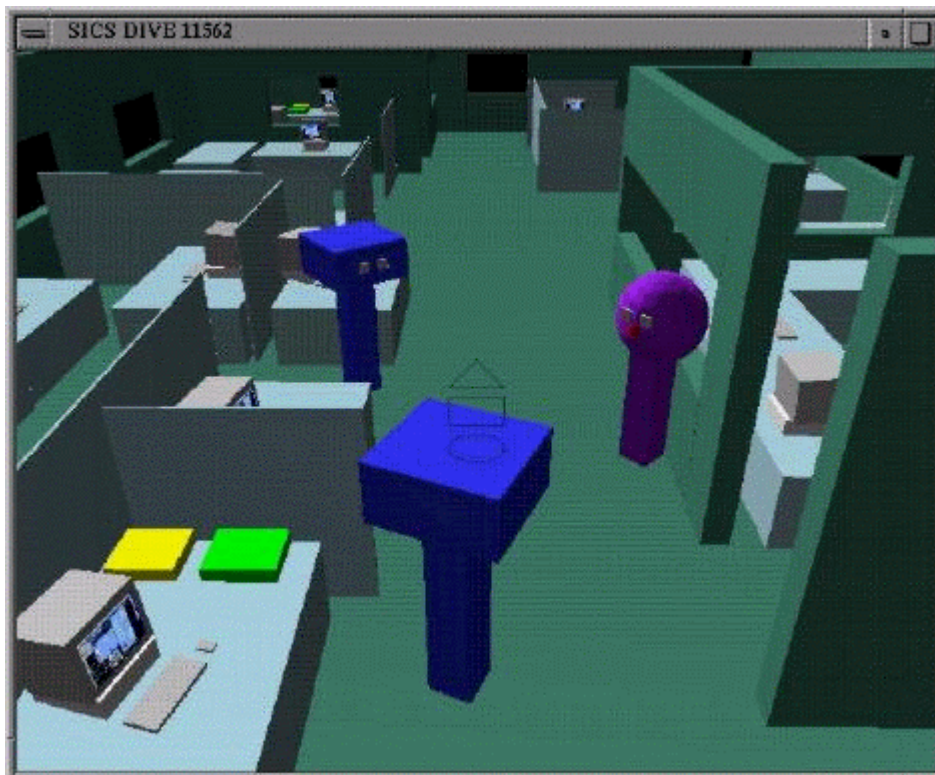


**Roger Young**  
**MSc IT: Software Engineering**  
**ICL Institute of Information Technology**  
**University of Nottingham**  
**Supervisor: Dr. Steve Benford**

***‘Designing a Group Navigation Tool for a  
Collaborative Virtual Environment’***



Submitted September 1996, in partial fulfilment of the  
conditions for the award of degree of MSc in Information  
Technology

## **Abstract**

With the growing research into Virtual Reality (VR), and in particular VR as a medium to allow groupwork and collaboration between distributed groups of people, there is a need for the development of a group navigation tool. A tool that lets a user-group move through a VR environment by their own actions and movements, and not by those of a single removed controller.

This paper therefore looks into meeting this requirement, and by using the principles of Computer Supported Co-operative Work (CSCW), shows how such a tool was researched and designed. A tool that monitors, and allows the monitored user-group to control the direction of their movement - either within an interactive movie, or a Collaborative Virtual Environment (CVE).

During development a secondary use for a variant of this tool was discovered. That of allowing a single user to change his or her view (not position) in the VR environment, by simply moving his or her head in the appropriate direction.

### **With thanks to:**

Dr. Steve Benford, Prof. Dave Elliman, and the Communication Research Group in the Computer Science Department, at the University of Nottingham. For all their help and advice.

# Table of Contents

<b>Chapter 1</b>	<b>4</b>
<b>1.1: Introduction</b>	<b>4</b>
<b>1.2: Introduction to Some Terms</b>	<b>4</b>
1.2.1: Computer-Supported Co-operative Work (CSCW);	4
1.2.2: Collaborative Virtual Environments (CVEs);	5
1.2.3: Why develop Collaborative Virtual Environments?	6
<b>1.3: Field of Research</b>	<b>7</b>
<b>1.4: Dissertation Plan</b>	<b>8</b>
1.4.1: Stage 1	8
1.4.2: Stage 2	8
1.4.3: Stage 3	8
<b>1.5: Plan Diagram</b>	<b>9</b>
1.5.1: Stage 1	9
1.5.2: Stage 2	9
<b>1.6: Plan Calculations</b>	<b>9</b>
<b>1.7: An Example Calculation</b>	<b>10</b>
<b>Chapter 2</b>	<b>12</b>
<b>2.1: Computer Vision</b>	<b>12</b>
<b>2.2: Image Understanding</b>	<b>13</b>
<b>2.3: The Image Understanding Process</b>	<b>13</b>
2.3.1: The Computational Approach	14
2.3.2: The Knowledge-based Approach	15
<b>2.4: Image Processing</b>	<b>15</b>
<b>2.5: Interviews with an Expert</b>	<b>15</b>
<b>2.6: Low-Level Image Processing</b>	<b>16</b>
<b>2.7: Digital Image Properties</b>	<b>16</b>
<b>2.8: Computer Hardware and Software</b>	<b>17</b>
2.8.1: The IRIX Operating System	17
<b>2.9: Image Handling Software</b>	<b>17</b>
2.9.1: ImageVision Library	17
2.9.2: Video Library	18
2.9.3: X Library	18
2.9.4: SGI File Format Library	18
<b>2.10: Camera Configuration</b>	<b>18</b>
<b>2.11: Libraries to be Included</b>	<b>19</b>
<b>2.12: SGI Image File Format</b>	<b>19</b>
<b>Chapter 3</b>	<b>21</b>
<b>3.1: Suitable Computer Languages</b>	<b>21</b>
<b>3.2: Program Specifications</b>	<b>21</b>
<b>3.3: Coding the Program Specifications</b>	<b>22</b>
<b>3.4: The Video Library (VL.H) Commands</b>	<b>22</b>

<b>3.5: Program 'grab.cpp' Pseudo-Code</b>	<b>22</b>
<b>3.6: Converting the Sample 'simplegrab.c' Code</b>	<b>23</b>
<b>3.7: Coding the Image Processing Algorithms</b>	<b>24</b>
<b>3.8: Program Developments</b>	<b>26</b>
<b>3.9: Creating the 'setup.cpp' Program</b>	<b>27</b>
<b>Chapter 4</b>	<b>29</b>
<b>4.1: 3-Dimensional Navigation</b>	<b>29</b>
<b>4.2: Program Requirements</b>	<b>29</b>
<b>4.3: Converting 'dgl_view.c' Code</b>	<b>30</b>
<b>4.4: Adding 'grab.cpp' Code</b>	<b>30</b>
<b>4.5: Creating the 'setup.cpp' Program</b>	<b>32</b>
<b>Chapter 5</b>	<b>34</b>
<b>5.1: 'dgl_view' Performance</b>	<b>34</b>
<b>5.2: Project Conclusion</b>	<b>34</b>
<b>5.3: Future Developments</b>	<b>35</b>
<b>5.4: Conclusion</b>	<b>35</b>
<b>References</b>	<b>36</b>
<b>6.1: Bibliography</b>	<b>36</b>
<b>6.2: SGI On-line Books</b>	<b>37</b>
<b>6.3: World Wide Web Pages</b>	<b>37</b>

# Chapter 1

## **1.1: Introduction**

This paper is my MSc in Information Technology dissertation, and looks at;

*'Designing a Group Navigation Tool for a Collaborative Virtual Environment.'*

With the growing research into Virtual Reality (VR), and in particular VR as a medium to allow groupwork and collaboration between distributed groups of people, there is a need for the development of a group navigation tool. A tool that lets a group of users (user-group) move through a VR environment by their own actions and movements, and not by those of a single removed controller.

During the project the requirements of such a tool were researched, as was any existing technology in this field. Once it was decided that some form of image capture and processing was to be used, various existing methodologies for computer vision were looked into. In the absence of any suitable algorithms for the image processing that was required, some were devised that could be later programmed.

The available hardware and software for the project was researched and a suitable program specification drawn up. During the coding and testing of the tool a secondary use for a variant of the tool was discovered. That of allowing a single user to change his or her view in the VR environment, by simply moving his or her head in the appropriate direction.

This single user program was added onto a Virtual Reality Modeling Language (VRML) viewer which then allowed a user to move in the 3-Dimensional world by using existing mouse functions, and then change the view of what he or she saw by moving his or her head. This made up the testing of the final program, because due to the time available it was not possible to add the program to a Collaborative Virtual Environment to test the user-group navigation tool. However the success of the single user tool demonstrated that the user-group tool would work technically, but not whether it could be used usefully by a participating user-group.

The paper is split into five chapters. The first chapter refers to the project background, planning and introduces some of the terms used. The second deals with the available resources, and the current theories and methodologies in the fields of computer vision and image processing. The third and fourth chapters cover the actual programming of the project, the decisions made and how the plans and theories from chapters one and two were applied. The final chapter concludes the project and looks at the success and possible uses of the final program. The paper includes a bibliography, and an appendix containing the code, and sample output of data and images captured while the program was running.

## **1.2: Introduction to Some Terms**

### **1.2.1: Computer-Supported Co-operative Work (CSCW);**

The following definition for CSCW, was taken from the WWW page [I];

*'CSCW aims at providing computer-based tools and techniques for the effective support of the collaboration of two or more humans in reaching a common goal or jointly performing a task. A simple classification of CSCW can be made with respect to the temporal and spatial relationship of the people involved in a collaborative task.'*

*Groupware is software and hardware for achieving CSCW. Specific hardware may include, e.g., audio or video equipment. Groupware software implements the shared work space in which two or more people can collaborate.'*

### 1.2.2: Collaborative Virtual Environments (CVEs);

The following definition of a CVE, was taken from the introduction to the CVE96 conference by Dr. Steve Benford;

*'A CVE uses distributed Virtual Reality technology to support communication among groups of people.'*

The following description for CVEs, was taken from the WWW page [II];

*'CVEs involve the use of distributed virtual reality technology to support group work. A necessary, but not sufficient, condition for a CVE is the provision of simultaneous multi-user access to a virtual reality system. However, there may be a world of difference between a multi-user system and one that actually supports cooperative work, and so a second condition is that the system must explicitly consider and support the needs of users who wish to work together.'*

*The essence of CVEs is that users are explicitly represented to each other within a shared space. Furthermore, they should be free to move around within this space, encountering each other and also objects and information of common interest. The interactive nature of true virtual reality systems means that they should also be able to interact with each other and with the objects and information.'*

*There has been a lot of development in CSCW in the past few years. Probably the most influential project in this area has been DIVE, a development at the Swedish Institute of Computer Science (SICS) which has resulted in a freely available CVE which has been used world-wide as the basis for a host of other projects [FahlÅn93, Carlsson93]. DIVE provides a general development environment for CVEs and more details can be obtained from Lennart FahlÅn (lef@sics.se).'*

*The development of effective CVEs has greatly increased the scope of CSCW, and it is now foreseeable that in the near future there will be experts separated by great distances, who will be working on problems together within a Virtual Environment (VE). This will simulate all the associated benefits of a real time meeting held within the same room. The experts will then be able to do anything that they could within a real meeting, ie. pick something up, adjust a diagram, talk to another specific member, and etc.. There is still a lot of work and research to do in this increasingly important field, including studying and*

*defining how groups actually act in real life situations, to allow for the translation of social aspects to a CVE.'*

### 1.2.3: Why develop Collaborative Virtual Environments?

The following reasons for developing CVEs, was taken from the tutorial on the WWW page [II];

*'There are several general reasons for supposing that CVEs might be useful. Three general ones are:*

*1. Support for natural spatial social skills - as inherently spatial creatures, human beings possess powerful spatial skills. The development of single user interface technologies has involved exploiting an individual's cognitive spatial skills - i.e. their ability to reason about space (e.g. spatial classification and navigation abilities). However, humans also possess social spatial skills whereby they make use of shared space as a means of negotiating interaction with one another. The importance of eye-contact in conversation, mingling at cocktail parties and behaviours such as queuing, jostling and scrumming are examples of how such skills are used. All of these are highly dynamic, subtle and adaptive behaviours and are a far cry from the kinds of "floor control" mechanisms that have previously been proposed for communication systems.*

*2. Inherent scalability - nearly all real-time CSCW systems (e.g. multi-media conferencing and media spaces) have focused on small scale interaction between a few concurrent users. There is a real need to address synchronous communication between many hundreds of thousands of simultaneous users. In order to do this, the underlying approach must have some inherent property that is scalable. Shared space is one such property as it provides a natural and intuitive way to interact with many people and yet separate what is immediate and local from what is peripheral and distant. Thus, one can talk to a few people nearby and yet see potentially thousands of others. Those who are sceptical about the possibilities or need for large scale interaction might like to spend a little time ruminating on the possibilities for communication, awareness and interaction within, say, a large sports stadium or at a large public performance (which may peak at around 150,000 simultaneous users).*

*3. Applicability to cooperative spatial tasks - some highly spatial tasks such as 3-D design (e.g. architectural design) and environmental planning which are currently the domain of single user virtual reality applications may also offer possibilities for collaboration.*

*The emergence of CVEs can be related to, and has been influenced by, several areas of technology.*

*First, a number of projects have been explicitly addressing the issue of cooperative work and virtual reality. These include the COMIC ESPRIT III Basic Research Action [Benford93] and the UK's Virtuosi project funded under the EPSRC/DTI CSCW Programme [Benford94b]. The former has been developing general techniques for CVEs, many of which are covered by this tutorial, whereas the latter has been focusing on the industrial application of these and*

*other techniques through two major pilots; the virtual factory for the manufacturing industry and the virtual catwalk for the fashion industry. Japanese efforts include Collaborative Workspace from NTT [Takemura92], an immersive system supporting real-time facial action capture and animation and also NTT's Interspace project which supports groups of people navigating a virtual city in order to go shopping and engage in recreational activities [Suzuki95]. The Greenspace project should also be noted as an example of a demonstration of a co-operative virtual space between the HITLab in the US and sites in Japan.'*

### **1.3: Field of Research**

This dissertation looks at the movement and interaction of a group of users, within an interactive movie or a CVE. At the moment most interactive movie and CVE action is dictated by a single controller, he or she decides how the scenes are going to pan out, the user is then carried along as a passenger. For future development in CSCW, it will become increasingly important for the users to be able to control the output by their combined actions (ie. they become a user-group). Simply this would mean that if most of the group started to focus on one area of the display then the focus of the main program would move to concentrate on this area. However natural human behaviour within groups allows the manipulation of a situation by individuals, people talk and persuade other people to join them in looking at something, the persuaders will then look at something else in return. The group is therefore not controlled purely by its movement, but also by the interaction of its members. This interaction should also be incorporated into the design of the application, as it is an important part of group behaviour, and therefore of CSCW.

The best existing examples of interactive movies are at Futuroscope, the French cinema and image theme park. Three of the attractions show some level of group controlled output.

1. The 360 Degree Screen; allows a group of people to take part in, for example, a scene from Le Tour de France, here there is one controller and the group remains stationary while the race moves around them.
2. The Simulator; has a group of people in a car and they are taken for a ride, for example, on a toboggan run with the car mimicking the trip down the slope. Again one controller dictates the action.
3. The Interactive Cinema; has a normal cinema screen with an audience, but the story is controlled by the audience pushing buttons to allow the majority to dictate how the story develops.

Individually these are early examples of group interactive cinema, but if the principle of (3) was incorporated into (1) and (2), you would have a proper group controlled interactive experience. In (1) and (2) for example it could be the movement of people inside the screen / car that dictates which way the focus of the trip would go. Some way of monitoring group movement and interaction could therefore help lead to more advanced interactive cinema. The same principles could also be translated into the newer field of CVEs, where an important use is for a group of people to hook up to one CVE server and move through the world as a user-group.

### **1.4: Dissertation Plan**

The aim of this dissertation is to look into a way of monitoring individuals who are making up a user-group, and who are interacting and moving - either within an interactive cinema or a CVE. The behaviour of the individual members of the group in both cases could be translated into a set of factors that could then be monitored by a program that then controls the output that the individuals see. This output would then be changed in accordance with the overall group's position, and the individual's movement changes. The ideal monitoring program would incorporate the following individual attributes, movement (in a 3-Dimensional plane), velocity, center of mass, proximity to others, weighting of any leaders' attributes, and also consider audio as well as visual output.

Obviously this would be too much for a three month project, and I will therefore be focusing on the movement of individuals within a user-group in a 2-Dimensional plane, ie. to monitor a group from above, thereby assuming that all the individuals will have a negligible difference in height.

There will be three distinct stages to this project:

#### **1.4.1: Stage 1**

Will be to create a program that will capture some sample image data of a similar form to the final data. The program will then be designed to convert and process the image, and then output to a file the centre of light density of the image in the form of co-ordinates.

#### **1.4.2: Stage 2**

Will be to expand the program so that it runs continuously grabbing and processing new images, also improving the speed of the program so that it is as effective as possible in a real time situation. The program will then be tested by using the movement of a monitored user-group. This information will automatically contain characteristics of the group, eg. people following someone else, co-operation between individuals, and etc..

#### **1.4.3: Stage 3**

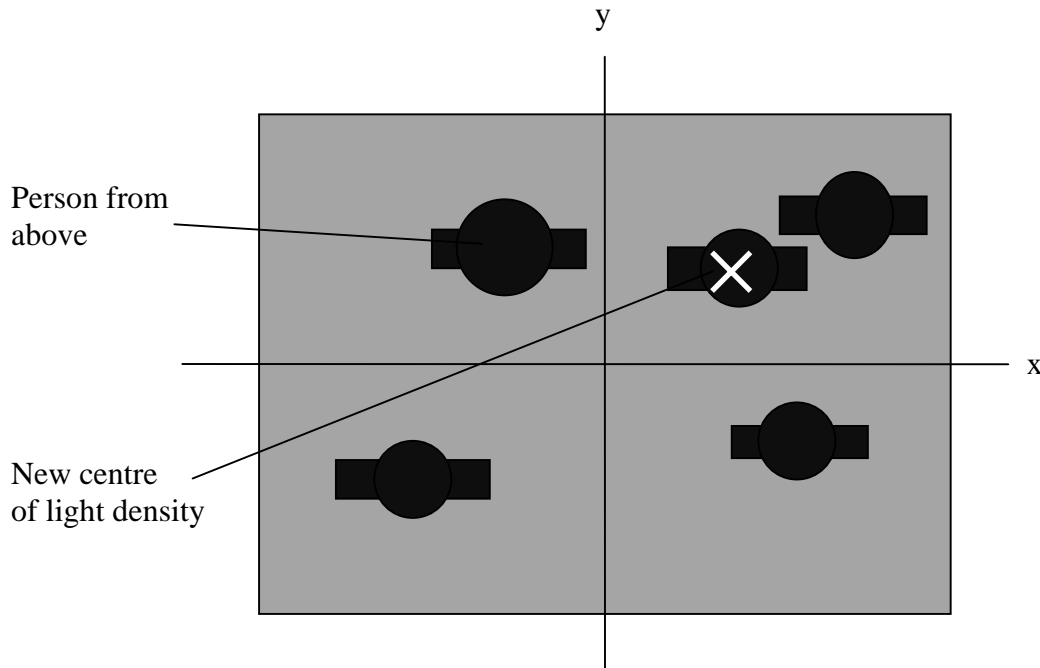
Will be to look at the type of input that would be accepted by a CVE, and trying to hook the program up to an existing, working CVE (in this case DIVE). Then a way will be found to change the program, in order to make the two compatible with each other. This will be the hardest test of the program, and may result in documentation of how to proceed with the research rather than a final working program.

## 1.5: Plan Diagram

### 1.5.1: Stage 1

Information in: normal image captured in the correct format.

Captured image of a user-group from above;



Information out: new centre of light density, calculated from the weighted light density of captured image.

### 1.5.2: Stage 2

Information in: realistic images continuously captured in real time.

[images as above, but continuously captured at run time]

Information out: new centre of light densities of the captured images, calculated from the weighted light density of each image at a time.

## 1.6: Plan Calculations

The program should capture an image, and convert this into an array of numbers.

An example of a digital image array (image size 6\*8) - for previous image

20	20	20	20	20	20	100	120
20	20	160	100	20	110	190	170
20	20	180	120	20	100	200	150
20	150	120	20	20	20	170	100
20	150	120	20	20	20	190	100
20	20	20	20	20	20	20	20

The program could then calculate the centre of light density by examining the luminance value and the distance from the centre of each pixel, and the total value of luminance for the whole image.

This could be achieved by using vectors, by using the centre of the image as the origin  $(0,0)$ , and adding on each pixels' vector values of  $i$  and  $j$ , times the luminance value ( $l$ ) of the pixel, divided by the total value of luminance for the whole image. The sum of all these values will give the  $x$  and  $y$  co-ordinates for the new centre of light density of the image (the negative and positive values of  $i$  and  $j$  mostly cancelling each other out).

The vector of the new centre of light density is  $(xi + yj)$ , where;

$$x = \text{sum}( l*i / \text{sum } l) \text{ for all } i$$

$$y = \text{sum}( l*j / \text{sum } l) \text{ for all } j$$

Therefore, the new centre of light density =  $(x,y)$ , where the origin =  $(0,0)$   
This means that the centre moves  $x$  to the right, and  $y$  up.

Where for an image of size  $480 * 640$ ;

$$-320 \leq x \leq 320$$

$$-240 \leq y \leq 240$$

This is an overall movement of magnitude;

$$m = \text{sqrt}(x*x + y*y)$$

At an angle  $z$ , where;  $\tan(z) = x / y$

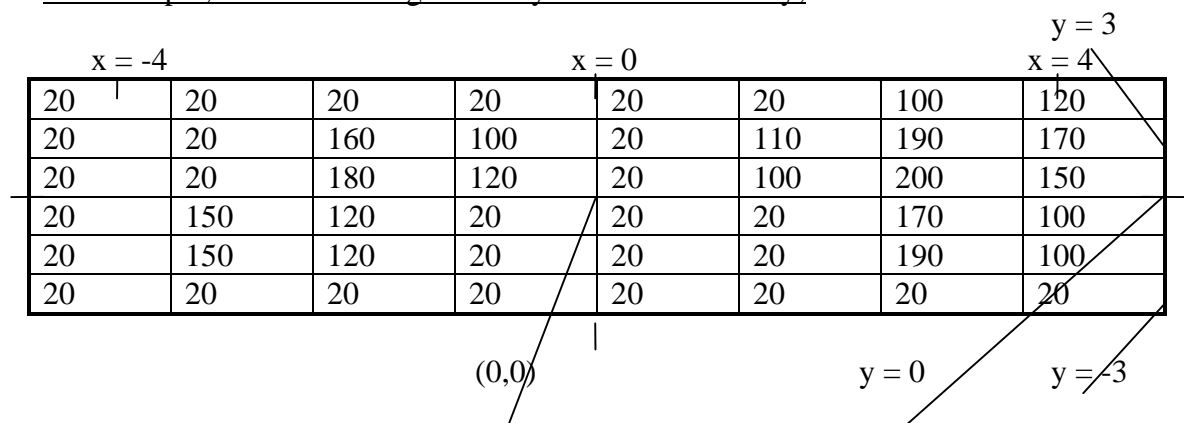
Therefore angle;

$$z = \text{arctan}(x / y)$$

ie.  $z$  degrees clockwise from the vertical, where  $0 \leq z < 360$

### 1.7: An Example Calculation

For example; the centre of light density would be found by;



Therefore;

$$x = \frac{-4(6*20) - 3(4*20+2*150) - 2(2*20+2*120+160+180) - 1(4*20+100+120) + 1(6*20) + 2(4*20+100+110) + 3(20+100+170+2*190+200) + 4(20+2*100+120+150+170)}{(28*20+5*100+110+4*120+3*150+160+2*170+180+2*190+200)}$$

$$= 0.828$$

$$y = \frac{+3(6*20+100+120) + 2(3*20+100+110+160+170+190) + 1(3*20+100+120+150+180+200) - 1(4*20+100+120+150+170)}{(28*20+5*100+110+4*120+3*150+160+2*170+180+2*190+200)}$$

$$\frac{-2(4*20+100+120+150+190) - 3(8*20)}{(28*20+5*100+110+4*120+3*150+160+2*170+180+2*190+200)} \\ = \underline{0.306}$$

The new centre of light density is at  $(\underline{0.828}, \underline{0.306})$

Converting this to percentages;

$$x = 0.828 * 100 / 4 = \underline{20.7\%}$$

$$y = 0.306 * 100 / 3 = \underline{10.2\%}$$

The new centre of light density weighted co-ordinate is at  $(\underline{20.7}, \underline{10.2})$

Note : This is in about the same position as the estimated centre from the sample image (1.5.1) the calculation was based on.

The magnitude of this co-ordinate is;

$$m = \text{sqrt}(20.7*20.7 + 10.2*10.2) = \underline{23.1}$$

The angle from the vertical is;

$$z = \text{arctan}(20.7 / 10.2) = \underline{63.8 \text{ degrees}}$$

## Chapter 2

### 2.1: Computer Vision

Jolion [4] discusses relevant methodologies in his paper ‘Computer Vision Methodologies’, in the paper he quotes a previous attempt to define the area of computer vision research.

*‘Trivedi and Rosenfield [(1989), On making computers “see”, ‘IEEE Trans. Systems Man Cybern’] proposed a classification of vision research as follows:*

*Vision is investigated by three different schools of the scientific community. Neurophysiologists attempt to understand how sensory and neural mechanisms of biological systems function. Perceptual Psychologists try to understand the psychological issues governing the task of perception, and Computer Vision Scientists investigate the computational and algorithmic issues associated with image acquisition, processing, and understanding.’*

There are some existing methodologies available for computer vision. One is by Marr and formulates [5, p.24];

*‘The different levels at which an information processing device must be understood before one can said to have understood it completely.’*

The three levels he is referring to are computational theory, representation and algorithm, and hardware implementation.

This methodology has been shown to be true not only in the specialised field of computer vision, but also for image processing in general. The three levels Marr [4, fig 1,p3] refers to are shown in the table on the next page.

#### Marr’s three levels of computer vision

Computational Theory	Representation and Algorithm	Hardware Implementation
What is the goal of the computation, why is it appropriate and what is the logical strategy by which it can be carried out?	How can this computational theory be implemented? In particular, what is the representation for the input and output, and what is the algorithm for the transformation?	How can the representation and algorithm be realised physically?

When designing a program that focuses on computer vision, there are some other methodologies available to follow. Some examples from Jolion [4] are; Durand’s Vision System, Mundy and Lavin’s a concept-based inspection system, and a general purpose vision system architecture.

There are basically two philosophies for solving computer vision problems, one is to follow or create a general system based on computer vision techniques - for example one of the above. The other is the ad hoc approach, this is based on the fact that computer vision is still a new field and consequently there are not yet reliable techniques available for all problems. It is therefore better for the designer to approach each case on its own merits, and try and find all of the components he or she needs in order to complete the task efficiently.

## **2.2: Image Understanding**

One of the more important goals of computer vision is image understanding, this means the constructing of an intelligent system that can extract information from an image. Traditionally the data involved is presented in a form that is easiest for a human operator to understand. This is particularly true for video footage. Human interpretative processes are still not well defined, and are therefore very difficult to simulate with computers. There has been no real development in designing a camera that displays images in a form that is easy for computers to interpret. Consequently, most of the computer vision techniques available are for computers to learn to understand existing image formats.

The Encyclopaedia of Artificial Intelligence [8] describes the purpose of image understanding as;

*'The goal of an image understanding system is to transform 2-D data into a description of the 3-D spatial temporal world: such a system must infer 3-D surfaces, volumes, boundaries, shadow, occlusions, depth, colour, motion.'*

Vernazza [7] analyses this statement and points out 'infer' and 'description' to be the key words in the definition. He gives the following descriptions of these words:

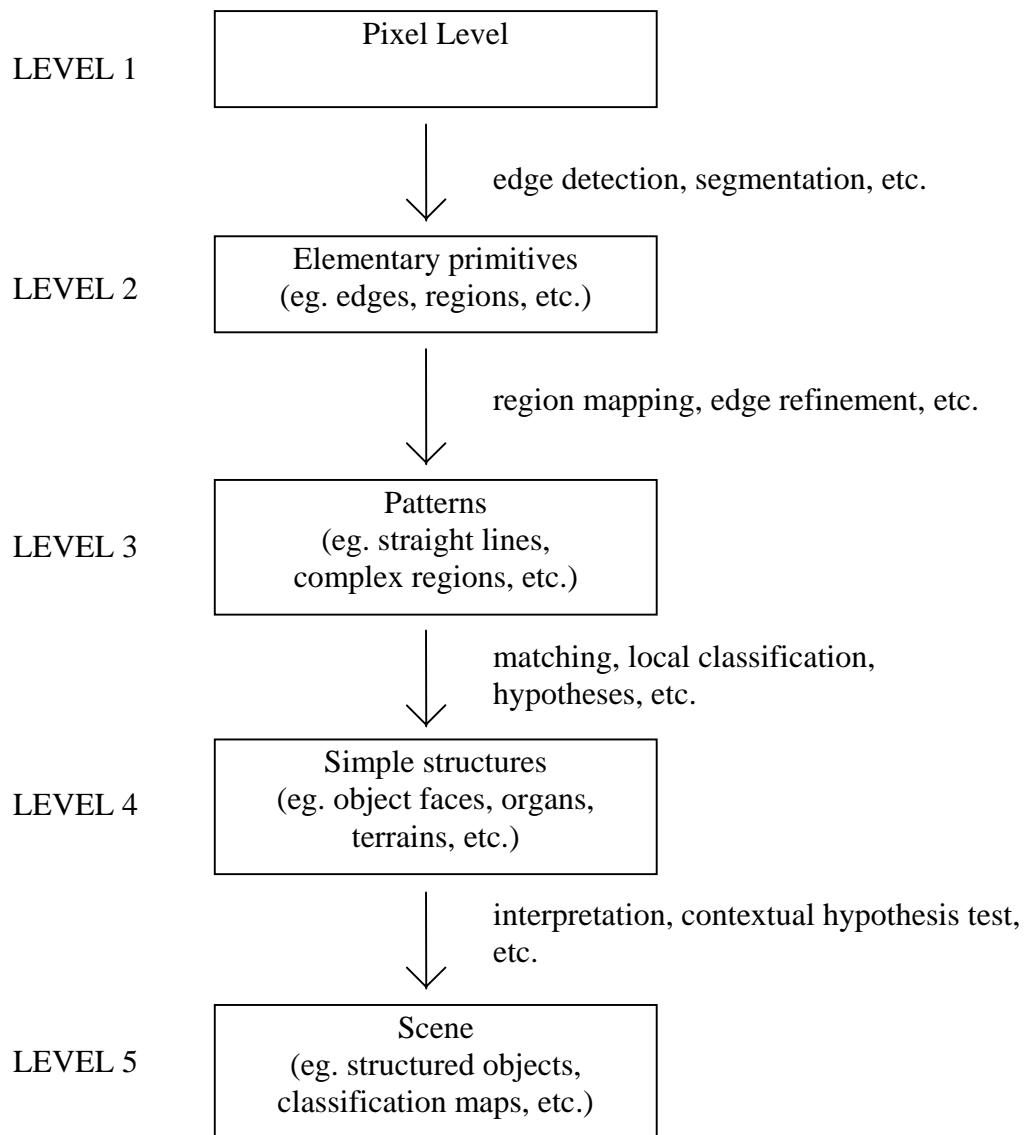
1. infer: *'the image acquisition process introduces a gap between input data and the world's properties, and such a gap needs to be bridged by some appropriate technique (eg. by using the general perspective theory, we can obtain a 3-D interpretation from the analysis of 2-D lines).'*
2. description: *'the result is a more or less detailed or abstract description of the world.'*

In this dissertation only a 2-D example will be looked at; however the above principles still hold true. The gap between acquiring the images and the required interpretation will be filled by the program which will attempt to extract the necessary information from the images. This information will be displayed in co-ordinate form, which will be a description of the focus of the user-group's position.

## **2.3: The Image Understanding Process**

The process refers to the development of systems capable of simulating human cognitive recognition. This process is very difficult. At present the process involves performing a set of progressive transformations on an image that lead to a final description of the image. Vernazza [7,Figure 3.1,p45] gives a;

Possible structure for the image understanding process:



EACH LEVEL HAS ITS OWN SET OF ATTRIBUTES

There are two main approaches to the problem of image understanding. These are the Computational Approach, and the Knowledge-based Approach.

### 2.3.1: The Computational Approach

Using this approach needs an understanding of the relevant image format in order to extract the necessary information. This leads to the construction of algorithms that recover information about the world from the image. This is done by mathematically studying the format of the image, and then making numerical algorithms that will process the image and give the desired information in a numerical form. This approach can lead to powerful recognition tools, but they are by nature very specialised and can rarely be general enough to cover many real situations.

### 2.3.2: The Knowledge-based Approach

This approach relies on the use of artificial intelligent techniques to attain three goals. These are representation of knowledge, the inference handling mechanism, and the control structure. The vision process relies on a lot of knowledge, and the main assumption of this approach is that each image can be explicitly represented in terms of this knowledge. This approach can lead to good general recognition tools, but this is not an easy task and it does not take account of the individual image format.

In the case of this dissertation the best approach to follow is the computational one. This is due to the recovery of information being both important and complex. The characteristics of a good computational approach are reliable mathematical tools, and a significant amount of information on the image format.

### **2.4: Image Processing**

A very important part of the program will be the input it accepts, before the program can be written or even designed the final form of this input must be detailed.

### **2.5: Interviews with an Expert**

In order to get some advice on the field of image processing I interviewed an expert in the field; in this case Prof. Dave Elliman. The first interview concentrated on the format of a digital image and a brief discussion of the plan from chapter 1, we also discussed possible languages that are suitable for image processing.

The following important development points transpired from the meeting.

1. It is possible to have a camera connected to a computer which could monitor the movement of the user-group detailed in chapter 1. The computer can be designed to take a snap shot every  $t$  seconds, and convert it into digital form. This could be in colour or black and white. Due to the length of the project and the nature of the program it would be better and faster to deal with only black and white images (ie. grayscale images). It would also be a good idea to invert the images (ie. black to white and vice-versa), making it possible to test for the concentration of white areas (white being the highest colour value, black the lowest).
2. It is important to get some realistic test data of the correct format before starting to write the program. This could be done by taking a photo and scanning it, then converting the image into the desired format.
3. There is a 'Visual Image Processing' (VIP) undergraduate course run by the Computer Science Department. There is a lot of software on the system for this course, which is used in tutorials to convert images into different forms, analyse images, and etc.. It could be possible to incorporate some of these programs, to help with the handling of the digital images by the monitoring program.
4. There are several good languages that cope with image processing, the visual languages such as Visual Basic can be used for this type of program, but they are not necessarily the most suitable. A traditional language such as C handles image processing very well, indeed the VIP course and its tutorial programs are written in C.

We later had a second interview to discuss the project and some of the image processing techniques I had been researching.

1. It was confirmed that a grayscale digital image was stored in a matrix, the co-ordinates of which are a representation of the position of each pixel on the real image. The array holds the value of the luminance / brightness of the image at each pixel point. Low-level processing of the image will therefore give all the necessary information in this simple case. The real image distances will be lost in conversion, but the scale is not necessary as the program will be using weighted values of distance to calculate the moments of the light density.
2. The possible cameras available for the project are a Sun Camera and an SGI camera (Silicon Graphics, Inc.), which can capture images in SGI (Silicon Graphics Image) or QuickTime format. It was recommended that I find out the file format of these images, either from the library or the Internet, to enable the designing of suitable algorithms for later coding.

Some of the terms discussed in the interview are explained here:

### **2.6: Low-Level Image Processing**

Sonka, Hlavac, Boyle [3,p.3] describe low-level image processing methods as;

*'Low-level methods usually use very little about the content of the images. In the case of the computer knowing image content, it is usually provided by high-level algorithms or directly by a human who knows the problem domain. Low-level methods often include image compression, pre-processing methods for noise filtering, edge extraction and image sharpening. Low-level image processing uses data which resemble the input image; for example, an input image captured by a TV camera is 2-D by nature, being described by an image function whose value is usually brightness depending on two parameters, the co-ordinates of the location of the image. If the image is to be processed using a computer it will be digitised first, after which it may be represented by a rectangular matrix with elements corresponding to the brightness at appropriate image locations. Such matrices are the inputs and outputs of low-level image processing.'*

### **2.7: Digital Image Properties**

Sonka, Hlavac, Boyle [3,p.30] describe a digital image as;

*'A digital image has several different properties, both metric and topological, which are somewhat different to those of continuous 2D functions with which we are familiar from basic calculus.'*

*'A digital image consists of picture elements with finite size - these pixels bear information on the brightness of a particular location in the image. Usually pixels are arranged into a rectangular sampling grid. Such a digital image is represented by a 2D matrix whose elements are integer numbers corresponding to the quantization levels in the brightness scale.'*

## **2.8: Computer Hardware and Software**

The Communications Research Group (CRG) are located in the Computer Science Department, as well as using a Sun Unix server, they also have SGI (Silicon Graphics, Inc.) workstations. The SGI workstations use the IRIX 5.3 operating system.

### **2.8.1: The IRIX Operating System**

In addition to standard Unix features, IRIX 5.3 provides software that supports the graphics and multi-processing capabilities of SGI platforms, including;

- Fast 3D graphics
- Parallel programming
- Real-time enhancements
- IRIS Graphics Library (IRIS GL)
- ImageVision Library (VL), an image processing library built on IRIS GL

## **2.9: Image Handling Software**

There are four types of development software on the IRIX system, which provide programming tools for handling images (contained in C libraries), which could be used on this project. These are the;

### **2.9.1: ImageVision Library**

Image processing applications start with an image consisting of pixel information stored in a file. These images can originate as a scanned photographic image, satellite data and numerous other sources. An image processing application can manipulate this image in ways that are meaningful to the user of the application.

The ImageVision Library (IL) is a set of tools designed for developers of image processing applications. The IL is written in C++, but has interfaces for C also. You can use this library to import, manipulate, display, and store images.

The ImageVision library contains objects and methods that allow an image processing application to;

- Import images created in a variety of different file types. The supported file formats include TIFF, GIF, and SGI.
- Process images using any sequence of the image processing operators supported by the IL. Image processing functions include colour conversion, arithmetic operations on pixel data, radiometric and geometric transformations, generations of statistical data for an image, spatial and non-spatial domain transformations, and edge, line, and spot detection.
- Display one or more images in an X Window. The IL provides many ways to control image displays, including stacking images or aligning them side by side, roaming a large image or doing a wipe to move one edge of an image to reveal what is stacked beneath it.
- Store processed images on disk.

### **Architecture of an ImageVision Library Application**

The IL implements an execution model that optimises memory usage and performance as image data is processed. This execution model

- supports the parallel processing features of silicon Graphics workstations
- caches image data to minimise file access
- is demand-driven so that only image data needed for output is processed

- chains operations together, which saves time because intermediate results don't have to be stored
- uses hardware acceleration of graphics operations whenever possible to improve performance of IL operations

The architecture can be represented as;

*Application <--> ImageVision Library <--> IRIS GL <--> IRIX*

### 2.9.2: Video Library

The Video Library (VL) is a collection of device independent C language calls for SGI workstations equipped with video capabilities. The VL includes generic video tools, including simple tools for importing and exporting data to and from current and future SGI products, as well as to and from third-party video devices that adhere to the SGI architectural model for video devices.

The VL works with other SGI libraries, such as OpenGL, the IRIS GL and the IRIS ImageVision Library.

### 2.9.3: X Library

The X library, known as *Xlib* is a C language programming interface to version 11 of the X Window System. This library enables a programmer to write applications with an advanced user interface based on windows on the screen, with complete network transparency.

*Xlib* provides an image structure that is capable of storing all the data corresponding to a screen area or pixmap. The major difference between an image and a pixmap is that an image is a structure on the client side, so its contents can be manipulated directly by the client, instead of solely through X protocol requests.

*Xlib* provides the routines *XgetImage()* and *XputImage()* that use the X protocol to transfer the contents of a window or pixmap into an image structure and to write the contents of an image structure back into a window or pixmap.

*Xlib* also provides a few minimal routines for manipulating image structures, including routines to create and initialise an empty image structure, destroy an image structure, get a pixel, set a pixel, extract a sub-image of an image, and add a constant value to all pixels in an image.

### 2.9.4: SGI File Format Library

This includes a library *libimp* which provides a C language Application Programming Interface (API) for reading and writing SGI format files, and for performing a number of format-independent image processing operations. These operations include colour space conversion and filtered image zooming. The library *libimage* includes the format of an SGI image (printed on the next page).

## 2.10: Camera Configuration

The camera available for this project are the SGI and Sun cameras, the latter of which provides the better images and is fully compatible with SGI image handling software. The camera can display images in;

- SGI, or QuickTime formats
- Movie or photograph formats
- Various types of compression (including variable jpeg, and uncompressed)

QuickTime is the Apple movie format. Kientzle [9] describes QuickTime movies by;

*'While it supports movie production very well, it was designed to support any type of time-based information. A QuickTime file can be as simple as a photograph.'*

SGI has no standard defined format for cross platform use, but it is compatible with all SGI equipment and software. Consequently the final program could run with some systems and not others. However the Communications Research Group's equipment is configured for using SGI, and a program using the SGI format would be compatible with all current and future CVEs used by the group. Also a program based on an SGI image would be able to use the SGI specific development tools already on the system. The SGI format is therefore probably the better choice for this project.

It should be possible to configure the camera for SGI formatted images, these can be of varying sizes from 160\*120 to 640\*480, and can be of movie or photographic format with jfif (jpeg) compression, or uncompressed.

The program can then be written using the image handling libraries, which would control the camera taking a picture every t seconds. The captured image would be stored in a buffer where it can be accessed by the program for digital conversion and image processing.

### **2.11: Libraries to be Included**

For this project it would be best to use the Video Library - this will automatically mean using the Graphics Library as well (see 3 above). The Video Library is detailed in the SGI on-line book [B, Part Three]. Included in this manual is a sample C program called 'simplegrab.c' which uses the VL to capture an RGB (red, green, blue) image from an attached SGI or compatible camera, display the image in a window on the screen, store the image in a buffer, and find a pointer to the information in this buffer. It should be possible to use this program as the basis for the project. Depending on the format of the stored image it may be possible to use the buffer information pointer to access the RGB values of the stored image. Or one or more of the other libraries could be used to convert the image into an array, convert the image array to grayscale, and process the array.

### **2.12: SGI Image File Format**

The `IMPImage` structure in the `libimage` library file contains the structure of an SGI image. It is detailed in the SGI on-line book [E, Appendix B, Page 2]. This structure in C is:

```
typedef struct _impImage
{
// Public image header information
ushort_t  imagic;    // SGI file magic number
ushort_t  type;     // Raster type (eg. verbatim,
```

```

// rle)
ushort_t dim; // Image dimension
ushort_t xsize; // X size
ushort_t ysize; // Y size
ushort_t zsize; // Number of channels (eg. RGB =
// 3)
long min; // Minimum intensity in image
long max; // Maximum intensity in image
ulong_t wastebytes; // Padding
char name[IMP_NAME_MAX+1]; // Image name
ulong_t colormap; // Image type
// (eg. colormap
// / normal)

// Private image header information
long file;
ushort_t flags;
short dorev;
short x;
short y;
short z;
short cnt;
short *ptr;
short *base;
short *tmpbuf;
ulong_t offset;
ulong_t rleend;
ulong_t *rowstart;
long *rowsize;
} IMPImage;

```

Note : *ushort\_t* and *ulong\_t* are unsigned short and long respectively.

Arguments:

*magic*; Magic number identifying file as an SGI Format file.

*type*; Bitwise - Or combined code indicating the raster encoding method and the number of bytes per pixel per channel.

*dim*; Number of dimensions to the image (colormap dim =1, grayscale dim = 2, RGB dim = 3).

*xsize, ysize*; Image size in pixels.

*min, max*; The minimum and maximum intensity values in the image (all channels combined).

*name*; A descriptive name string for the image.

*colormap*; The image type.

## Chapter 3

### **3.1: Suitable Computer Languages**

An important part of the program design will be selecting a suitable language in which to write the program. Important factors to take into account will be the language's capabilities at; handling the image processing, and being not too difficult to learn within the project's time frame.

It is also important to consider what the monitoring program is going to be used for, ie. just to research the subject and lay down guidelines for someone else to follow if the project meets its objectives, or for someone to later convert the program and add it onto an existing CVE.

The program is being designed with collaboration from current CSCW projects run by the Communication Research Group (CRG) at the University of Nottingham, which uses the DIVE CVE - which is written in C and runs on a multi-user Unix operating system. The project will use some of the CRG's hardware such as the SGI or SUN camera. The SGI image processing libraries are in C, as are the Visual Image Processing course programs, and run on Unix. The program would therefore be of the most use to the potential end users if the program was compatible with the current system, ie. if it was in a C-based language written on a multi-user operating system.

In particular the SGI system provides a C based library called VL.H, to be used by developers in order to control the camera, and for image handling This library allows the user to configure the camera and image settings, and handle a buffer that stores the captured images.

It would therefore be best to write the program in C++ (which is one of the languages I am most familiar), on a Unix operating system (which is by nature multi-user). It would then later not be too difficult to convert the final program, into an add-on application to a DIVE CVE.

### **3.2: Program Specifications**

The program needs to accept as input images of the user-group which it will continuously capture. It will then convert these images into a digital data form and process the digital data to find the centre of light density, magnitude and angle. These calculated results will then be written to a file, and to the screen if requested. A set-up program will be created so that the user can specify the program options before running the main program.

The program will be built around the basic functions of the sample program 'simplegrab.c' discussed in chapter 2, this program finds any attached cameras, then captures an image with the camera, displays the image in a window, stores the image in a buffer (in an SGI format file, also detailed in chapter 2), and finds a pointer to the image RGB information contained in the buffer.

The sample program will need to be changed so that the main image grabbing, displaying, storing and data transfer functions are contained within a loop. The pointer to the RGB information is used to transfer the RGB values into temporary variables, which are converted to grayscale and inverted, and then stored in an array.

The program should then process the digital image array using the algorithms from chapter 2, and store the calculated results of each processed image in a linked list. Also printing the list to a file when the program is terminated, and onto the screen during run-time (if requested).

Finally if it is possible within the project time frame there may be an attempt to hook up the program to a working CVE, and then do some testing and analysis of how successful the program is at navigating in a 3-Dimensional environment, and what needs further development.

### **3.3: Coding the Program Specifications**

Including the SGI C libraries adds the following commands to the developer's tools.

#### **3.4: The Video Library (VL.H) Commands**

*svr* : the connection to the video  
*path* : the path between the camera and the memory  
*video buffer* : the buffer that collects the incoming images  
*val* : set the control values for the video card  
*info* : set of information about the incoming picture

#### **3.5: Program 'grab.cpp' Pseudo-Code**

The final program will need the following declarations and functions:

include video library : to use SGI commands  
include graphics library : to use SGI commands  
include maths library : to code image processing algorithms  
open\_file : open output file  
declare data\_structure : create a linked list to hold results  
declare image\_array : create an array to hold the converted digital image  
declare IRIS GL commands : create the functions and variables listed above (3.4)  
declare pointers : create global pointers for the linked list

start\_of\_main : start of main program  
start\_output : sent header output to file (and screen if requested)  
open\_video : open the video connection  
create\_buffer : create a buffer in which to store the captured images  
error\_handler : check output file, buffer, video connection opened properly  
begin\_loop : start loop  
    data\_transfer : capture an image and transfer to memory  
    show\_image : display the captured image in a window  
    get\_image : copy the image from memory into a grayscaled, inverted  
                image array  
    process\_image : process the image array, finding center of light density  
                    of the image  
    convert\_angle : find the angle of the center of light density, and  
                    convert it to degrees  
    add\_data\_structure : add calculated results to linked list  
    print\_output : print output to screen  
    reset\_buffer : clear buffer  
    check\_status : test for end of loop condition

end\_loop : exit loop  
 close\_video : close the video connection  
 print\_data\_structure : print linked list data to file  
 remove\_data\_structure : delete linked list  
 close\_video : close video connection  
 close\_file : close output file  
 end\_of\_main : end of main program

### 3.6: Converting the Sample 'simplegrab.c' Code

The sample image grabbing program previously described is written in C. It is not procedurised, and does not contain a program loop. It executes once stopping to give the user time to look at the grabbed image in a window. Once the user hits 'enter' the program terminates - after deleting all paths and closing the video connection. The first programming step was to procedurise the program, splitting the existing code into the functions described in the pseudo-code. The error checking function was also created at this stage. The next step was to add the output file and the linked list, including functions to add, print and remove items from the list. During this stage the program was converted from C to C++. Next it was necessary to create a program loop that would make the program run continuously until prompted by the user to stop. This required extra VL functions to be used, namely `vlResetBuffer()` which cleared the buffer at the end of the loop, ready to have a new image captured.

The program now continuously captured and held in a buffer images of the format previously described. It was then necessary to extract from the buffer (using the information pointer already created) the RGB details.

After some refinement in order to save memory - namely including within one loop the code that copied, grayscaled and inverted the image, and using temporary variables instead of arrays to store the RGB details. The image RGB details were copied into three temporary variables, by increasing the subscript value of the information pointer (`dataPtr`). This pointer subscript value initially was calculated from the current *x* and *y* co-ordinates (*i* and *j* in the loops), and the window width. The RGB values were then grayscaled (ie. single values of luminance from 0 to 255 - for an 8 bit image), using the matrix equation from the SGI on-line book [E, Appendix B, p8]:

$$\begin{bmatrix} 0.299 & 0.587 & 0.114 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} W \\ W \\ W \end{bmatrix}$$

By the taking this value from 255 the image is inverted, the newly grayscaled and inverted image was then copied into the global digital image array.

The code fragment that converts the image is:

```

// Copy RGB details from image in buffer into three
// colour arrays
for(j=0;j<ysize;j++)
{
    for(i=0;i<xsize;i++)
    {
        // Get ptr to RGB values in buffer
        ptr = i*4+j*xsize*4;
    }
}

```

```

// Copy RGB details from buffer into temporary
// short values
bbuf = dataPtr[ptr+1];
gbuf = dataPtr[ptr+2];
rbuf = dataPtr[ptr+3];

// Convert RGB details to grayscale and insert
// image into array
digital_image[i][j] = (int)(0.299*rbuf +
    0.587*gbuf + 0.114*bbuf);

// Invert grayscale image in array
digital_image[i][j] = 255 - digital_image[i][j];
}
}

```

### 3.7: Coding the Image Processing Algorithms

The image was now held in an array, where the x and y image co-ordinates of each pixel was represented by the address of the pixel in the array, and at the pixel's address was held the value of the luminance of the image (0 to 255) at that point. The image could now be processed.

The processing was done by coding the algorithms from chapter 1. In order to use the square root and trigonometric functions the Maths Library (math.h) was used.

The equations (from chapter 1) to find the co-ordinates of the center of light density of the image, and the magnitude and angle of these co-ordinates are:

Co-ordinates;

$$x = \text{sum}( l*i / \text{sum } l) \text{ for all } i$$

$$y = \text{sum}( l*j / \text{sum } l) \text{ for all } j$$

Co-ordinates as percentage values;

$$x = x / \text{xsize} * 100$$

$$y = y / \text{ysize} * 100$$

Magnitude;

$$m = \text{sqrt}(x*x + y*y)$$

Angle;

$$z = \text{arctan}(x / y)$$

In order to find the co-ordinates of the center of light density the total luminance value for the whole array needed be calculated. This was done by the following code (the max and min values of l were also found to help with setting up the camera in a practical situation), the following fragments are from the process\_image function:

```

// Find total luminance of array
for(j=0;j<ysize;j++)
{
    for(i=0;i<xsize;i++)
    {
        // Find the total value of luminance

```

```

ltotcalc = ltotcalc + digital_image[i][j];

// Find max value of luminance
if(digital_image[i][j] > lmaxcalc)
{
    lmaxcalc = digital_image[i][j];
}

// Find min value of luminance
if(digital_image[i][j] < lmincalc)
{
    lmincalc = digital_image[i][j];
}
}
}

```

The program could now calculate the co-ordinate value of the center of light density. It did this by finding the distance of each pixel point from the origin (0,0). The origin was found by splitting the image into four equal parts, the middle point being the origin. In order to have even distances that will cancel out and not leave an error value in the answer, the origin was placed between values of i and j (ie. the x value of the origin has xsize/2 on either side of it). This was achieved by the value of  $i - xsize/2$  moving from being -1 to +1. Zero was therefore skipped by adding on a weighting value w1 (this value is 0 for  $i < xsize/2$ , and 1 for  $i \geq xsize/2$ ). The same was done for the j value, but this goes from positive to negative. The following fragment of code finds the co-ordinates of the center of light density:

```

// Find co-ordinates of the center of light density
for(j=0;j<ysize;j++)
{
    for(i=0;i<xsize;i++)
    {
        // Add weighting w to remove (0,0) co-ordinate
        if(i >= xsize/2)
        {
            w1 = 1;
        }
        else
        {
            w1 = 0;
        }
        if(j == ysize/2)
        {
            w2 = 1;
        }

        // Find luminance times distance
        ltimesx = digital_image[i][j] * (i + w1 -
            (xsize / 2));
        ltimesy = digital_image[i][j] * ((ysize / 2)
            - j - w2);

        // Find sum of weighted l times d's
        xcalc += ltimesx / ltotcalc;
        ycalc += ltimesy / ltotcalc;
    }
}

```

```
}
```

The co-ordinate values were then converted to percentages, so that the differing values of the window height and width do not mislead a user (or CVE) who reads the output. The following code fragment does this:

```
// Convert (x,y) to percentages
  xcalc = xcalc * 100 / (xsize / 2);
  ycalc = ycalc * 100 / (ysize / 2);
```

The magnitude of these co-ordinates is found by this code fragment:

```
// Find magnitude
  magnicalc = sqrtf((xcalc*xcalc) + (ycalc*ycalc));
```

The angle of these co-ordinates is found from the following code (taken from the `convert_angle` function):

```
// Find inverse tan of angle in radians
  anglecalc = atanf(xcalc/ycalc);
```

This angle was in radians and did not take account of which segment in the image the co-ordinates were from. For example if x is negative and y positive, the angle given would be the same as for x positive and y negative. The program therefore converts each angle to degrees and then tests each case of x and y before adding a constant value (if needed) to convert the degrees to 0 to 360, the code fragment is:

```
// Convert angle to degrees, times by 180 and divide by
// pi
  anglecalc = anglecalc * 180 / 3.14159;

  // Convert angle to 360 degrees
  if(xcalc > 0 && ycalc > 0)
  {
    anglecalc = anglecalc;
  }
  else
  {
    if(xcalc < 0 && ycalc > 0)
    {
      anglecalc = 360 + anglecalc;
    }
    else
    {
      anglecalc = 180 + anglecalc;
    }
  }
}
```

### **3.8: Program Developments**

Once the program was running and being tested, I realised that another use for the program would be to allow using it with a camera it in a horizontal position. This would then monitor the user-group to see if they moved to the left or right. This would be an easier way to use the program when time to set up a room for overhead shots is limited. This would be very useful when testing the program.

After a discussion with Dr. Steve Benford it was realised that this would be particularly useful when monitoring a single user, the camera could be placed on top of the screen and it could monitor the head movement of the user.

In order to convert the program a new output standard was needed. If the y co-ordinate was made zero then the output would be of the form  $(+or-x, 0)$ . If the value of x was negative then the head would of moved to the right (as the captured image is reversed on the screen), and the angle would therefore be 270 degrees. A line of code needed to be inserted to cancel the y co-ordinate if the horizontal camera position was selected. The following line of code was inserted after the co-ordinates, and before the magnitude and angle were calculated (in the `image_process` function):

```
// Cancel y co-ordinate for horizontal camera position
if(CAMERAPOSN == 'h')
{
    ycalc = 0;
}
```

The co-ordinates and magnitude were automatically output correctly, but the angle needed to be worked out differently as  $x/y$  (for  $y=0$ ) would give infinity. The following code was inserted at the start of the `convert_angle` function, before the arctan was calculated (this arctan value is then no longer calculated if the camera is in the horizontal position):

```
// Find angle for horizontal camera position
if(CAMERAPOSN == 'h')
{
    if(xcalc > 0)
    {
        anglecalc = 90;
    }
    else
    {
        anglecalc = 270;
    }
}
```

### **3.9: Creating the 'setup.cpp' Program**

In order to give the user a chance to input the setup options that arose while planning and programming, a setup program was also written. The user could run this program before compiling the main grab program and then select whether the camera was in the horizontal or vertical position, to output to the screen during run time or not, and the size of the image to capture. This final option was included for the use of later developers, all the code was written so that if a different image size is grabbed all the program declarations and algorithms will not need altering. However in order to grab a different image size, the sample program code that formed the base of the program would need altering.

The setup program therefore prompts the user to input the setup options desired, the program then writes a new header file 'global.h' which is included by the 'grab.cpp' program when it is compiled. The user can then compile two separate programs one for a horizontal, and one for a vertical camera. The programs will be identical except

that they will run differently for each camera position (eg. by compiling one 'grab-v' the other 'grab-h'). The setup program would have been run before each was compiled. The setup program also displays the command line entry needed to compile the selected camera position program. This allows the user to copy it to the command line and compile the program.

The following is an example of the setup program running:

```
desire%./setup
Input setup configuration information for 'grab' when
prompted.
-----
Input destination of output file (f), or screen and file
(s) : s
Input the camera position horizontal (h), or vertical (v)
: v
Input the height of the image (480) : 480
Input the width of the image (640) : 640
You are now ready to run 'grab', at the command prompt
input
%g++ grab.cpp -o grab-v -lvl -lgl -lm<enter> (to compile)
%./grab-v<enter> (to run)
```

This run of the setup program will create the following 'global.h' file (which is then included in the main 'grab.cpp' program):

```
// Data input from setup for 'grab'
// Global definitions and constants
//-----
// Define true and false
#define TRUE 1
#define FALSE 0
// Global constants
const char OUTDEST = 's';
const char CAMERAPOSN = 'v';
const int WIN_HEIGHT = 480;
const int WIN_WIDTH = 640;
```

## Chapter 4

### 4.1: 3-Dimensional Navigation

The final stage of the project was to look at ways of navigating in 3-Dimensional (3D) worlds, for example; a CVE or a VR created model. Models can be written in the Virtual Reality Modelling Language (VRML), which is specifically for making 3D models of everyday things or imaginary objects and places. The VRML creates geometrical shapes (squares, spheres, etc.) to give the objects perspective and depth when viewed on a 2-Dimensional screen. These 3D worlds can be navigated by the user using the mouse and the keyboard to;

*fly* : fly towards or away from the object by moving the mouse up or down.

*walk* : move the view to the left or right and closer or further away by moving the correct mouse left or right and up or down respectively.

*rotate* : rotate the object clockwise or anti-clockwise by holding down the correct mouse button and moving in the corresponding direction.

*reposition* : move the object left or right and up or down, by using mouse movement and the correct mouse button.

These principles are used in the navigation of the live3D (extension .wrl) files displayed in the latest version of the Netscape browser. These files are written in the Virtual Reality Modelling Language (VRML). These principles have also been used by the CRG to navigate the same VRML files as test scenarios for navigating a CVE.

A good test of the grab program would therefore be to use it to navigate a 3D model. Due to the time constraint it would be sufficient to use the horizontal camera positioned program, to monitor the head movement of a single user and change his or her view of the 3D object correspondingly, without changing his or her position (this will still be done by using the mouse controls).

If the program panned the view round to the left (ie. moved the object left across the screen) when the head was moved to the right and vice-versa, it would simulate someone looking into the corner of an image to get a view of what was there. This would be a more realistic way of behaving for the user, and therefore make the program easier to use.

### 4.2: Program Requirements

The CRG already had a 3D viewer program called 'dgl\_view.c' written in C that let the user navigate a 3D world by using the mouse and the keyboard. This program should be converted to incorporate all of the necessary grab code, and then monitor the head movement of a single user and move the view of a 3D object in the correct direction. This will require the movement of the user's view around an axis in 3D. The other two axis should be set to zero as the view is only being rotated in one plane. The program should also still allow the user to change his or her position in the 3D world by using the mouse and the keyboard.

In order to convert the viewer program to allow a user-group to navigate a 3D world, the program must be converted to not change the view displayed, but to actually change the user-group's position in the 3D world. This could be achieved by making one axis zero, and repositioning the plane of the other two axis around the first axis by a function of the magnitude at the angle found by the grab program, with the camera in a vertical position.

The results from a 3D object in the viewer would not make much sense to a user-group navigation. This really needs the viewer to display a 3D world, or be hooked up to a CVE. Due to the time frame of the project, the grab program will be tested by a single user using the horizontal camera position. This will however show whether or not the user-group navigation version of the grab program is technically correct and possible.

#### **4.3: Converting 'dgl\_view.c' Code**

The 'dgl\_view.c' program was written in C, while C++ is backwards compatible and it should be possible to recompile the viewer program as a C++ program. It was a good idea at this stage to convert the grab code to C as it was copied over. This was not too difficult as mainly just the output to the screen and file were done using C++ functions, and the converted viewer did not need this output as it would have only slowed down the navigation of the 3D world. The main program algorithms then compiled with a C compiler, without needing much altering.

The viewer program had a timer callback function that when called checked the state of the mouse buttons and the keyboard, this function needed to be changed to include a check on the output from the grab code for a change of the user's head position.

The main program calls the timer function, and before this was done the program needed to open the video connection and create the buffer. After the loop had been exited the program needed to tidy up, closing the video connection and deleting the buffer.

#### **4.4: Adding 'grab.cpp' Code**

The timer\_callback function needed the following code to be included after the mouse events were checked. The grab functions called by this function were defined before they were called.

The pre-defined (in a header file created by the CRG) function was called;

```
dgl3_quat_from_euler(&yaw_delta,0,current.angle,0)
```

This changed the angle about the y axis while keeping the x and z axis the same (ie. 0). The current angle sent to the y co-ordinate was the angle returned from the head movement of the user using the horizontal camera position (the final converted viewer code allows the use of either camera position, defined during set-up). Since this angle (not converted from radians to make it acceptable by the quat function) is either  $+ or -\pi/2$  (90 degrees) it would have moved the user's view a quarter of the way around to the right or left, therefore four movements to either side would bring the user back to the starting position. This was too large a step so the angle was divided by 3 as it was calculated. This moved the user's view by either  $+ or - \pi/6$  (30 degrees), this gave a more realistic and useful movement of the view of the user.

The grab code was included in an else statement. This meant it was only called if there were no mouse events in that call of the timer. This is to stop the user getting confused by multiple output. The grab code also includes a counter so that it is called only once in every five non-mouse timer events. This was due to the fact that the timer is called at a rate of  $1000/(frame\_rate)$ , this was fine for mouse events, but the grab code doing image processing needs more time.

The `draw_frame()` function at the end of the timer\_callback function is what actually draws the changed frame, by using the changes previously defined within the function. The following code is for the timer callback function.

Note: the mouse events were unchanged and therefore not included.

```
void
timer_callback(crg_timer_p timer, void *user_data)
{
    dgl_ordinate_t y_factor, x_factor;
    /*
       If the mouse buttons are pressed, then handle
       navigation
    */
    if (button_state.state[0] ||
        button_state.state[1] ||
        button_state.state[2]) {
        MOUSE EVENTS (NOT INCLUDED HERE)
    }

    else {
        if(counter == 5) {
            dgl3_quat_t yaw_delta;

            /* reset counter */
            counter = 0;

            /* Transfer image data to buffer */
            data_transfer();

            /* Add output to structure */
            current.frame++;

            /* Copy, grayscale, invert image into an array */
            get_image();

            /* Process image */
            process_image(&current);

            /* send frame no and angle to screen */
            fprintf(stderr, "Frame number = %I : ",
                current.frame);
            fprintf(stderr, "movement angle %f\n",
                current.angle);

            /* send head movement to dgl */
        }
    }
}
```

```

        dgl3_quat_from_euler(&yaw_delta, 0, current.angle,
0);
        dgl3_quat_mult(&(vp->pos.rot), &(vp->pos.rot),
&yaw_delta);

        /* Reset buffer */
        vlBufferReset(svr,buffer);
    }
    counter++;
}

draw_frame();
}

```

Note: The OpenGL code to display the captured image in a window was commented out at this stage. This was to avoid any clashes with code from the 'dgl\_view.c' program that used a later version of GL to display the 3D model.

The following code was included at the end of the main program, the initialisation functions being called before *crg\_select\_loop()*. The functions to tidy up were called after this function.

```

/* set up timer events to draw frames */
timer = crg_timer_new(1000/frame_rate, TRUE,
timer_callback, NULL);

/* Open the video connection */
open_video();

/* Create new buffer */
create_buffer();

/* initialise frame counters */
counter = 0;
current.frame = 0;

/* enter main loop, handling timer and interaction
events */
crg_select_loop();

crg_timer_delete(timer);

/* Close the video connection */
close_video();

/* delete the primitives */
dglr_primitive_delete(r_prim);

/* close the window */
dglr_drawable_delete(drawable);

```

#### **4.5: Creating the 'setup.cpp' Program**

As with the grab program, 'dgl\_grab.c' can be used by a single user or by a group of users with the camera horizontal or vertical respectively. The setup program is therefore similar but it does not contain the output options, it also gives the new compilation and running commands. The program is compiled with a 'Makefile'

called by inputting 'gmake' at the command line. There are a lot of extra libraries included to compile the converted viewer program, and using a makefile makes it easier to administer a directory. It will automatically pick up any changes to any of the programs that make up the final executable program file.

## Chapter 5

### 5.1: 'dgl\_view' Performance

The last chapter involved the adjusting of the VRML viewer code to allow a user to change his or her view in a 3D VRML world. The testing of this program was the bulk of the testing for the project. Following is a report on the effectiveness of the viewer program;

*'The dgl\_grab program performed well during testing, it was possible for the user to position himself or herself within a 3D VRML world by using the mouse, and then to let the grab code monitor any head movements, and change the user's view correspondingly. Therefore by leaning left or right the user could control an object's movement across the screen without having to touch the mouse. Although the dgl\_grab code worked at least twice as quickly without the output information of the initial program, the movement was still slow and jerky. Running it on a faster computer would probably help the user have more immediate feedback, and therefore make it easier to use. It would probably be useful to also try making minor adjustments to the angle sent to the viewer, in order to try and get the best and most realistic movement. It would also be useful to include an area in the middle where if the user kept his or her head upright their view in the 3D world would not change. This would only need a simple if statement to be added in the timer\_callback function, and some trial adjustments would soon reveal the best values for the boundaries of the middle area.'*

The viewer program and the report details were tested on another user, who confirmed that this analysis was reasonable.

### 5.2: Project Conclusion

The project finished with three executable programs:

*grab-h;* a program that monitors a single user and outputs to a file and screen (if requested), the left or right movement of the user, and the corresponding angle (see appendix I.1 and II.7).

*grab-v;* a program that monitors a user-group from above and outputs to a file and the screen (if requested), the angle and magnitude that the focus of the user-group has moved in (see appendix I.2 and II.6).

*dgl\_grab;* a program that monitors a single user and changes his or her view in a 3D VRML viewer, corresponding to the user's head movement (see appendix I.3 and III.6).

There are also the two setup programs that write the 'global.h' header file for the two grab and the dgl\_grab programs. These allow the user to input the functionality required from the final executable. They also include the commands necessary for the programs to be compiled; in the case of the dgl\_grab program this is via a makefile, which is also included.

All these programs and all their earlier versions have been handed to the CRG for inclusion into their CVE and 3D viewer software archives. A diskette with the programs is not included with this paper due to the programs being written specifically for use by the CRG, and not being compatible with anything but quite specialised hardware and software (detailed earlier).

### **5.3: Future Developments**

The obvious development will be to combine the grab and dgl\_grab programs to make a program that will allow a user-group to navigate a 3D VRML world (as detailed in the program requirements of 4.2). Once this is achieved and tested the grab program could be added onto a CVE, and tested with a user-group for the practicalities of using such a tool.

The single user version of the grab program could also be added to a CVE, the practicality of a user navigating a 3D world without a headset could be experimented with by doing this.

### **5.4: Conclusion**

The project on the whole was very successful, even though the overall aim of the project of allowing a user-group to navigate a CVE was not achieved. This was due to the time frame of the project, which made it impossible to test the final program in this environment. The above report on testing the single user version of the program shows that the user-group navigation tool will work technically. However it does not show whether or not the navigation tool could be used usefully by a user-group in a practical situation. The converting and testing of this program with a working CVE is another project in itself.

## References

### 6.1: Bibliography

1. Loeffler, C.E., Anderson, T.,(Eds), (1994), '**The Virtual Reality Casebook**', Wiley Professional Computing.
2. MacDonald, L., Vince, J., (Eds), (1994), '**Interacting With Virtual Environments**', Van Nostrand Reinhold.
3. Sonka, M., Hlavac, V., Boyle, B., (1993), '**Image Processing, Analysis and Machine Vision**', Chapman & Hall Computing.
4. Jolion, J-M., (1993), '**Computer Vision Methodologies**', CVGIP (Computer Vision Graphics & Image Processing) Volume 59 No. 1 (1995).
5. Marr, D., (1982), '**Vision**', Freeman.
6. Sommerville, I., Bentley, R., Rodden, T., Sawyer, P., (1994), '**Cooperative Systems Design**', The Computer Journal, Volume 37 No.5 (1994).
7. Vernazza, G., (1991), '**From Numerical to Symbolic Image Processing: Integration of Computational and Knowledge-based Approaches**', Computer Vision: Craft, Engineering, and Science, Vernon, D.,(Ed), Workshop Proceedings.
8. Shapiro, S.,(Ed), (1989), '**Encyclopaedia of Artificial Intelligence**', J. Wiley.
9. Kientzle, T., (1995), '**Internet File Formats**', Coriolis Group Books.
10. Bourne, S.R., (1983), '**The Unix System**', International Computer Science Series.
11. Schildt, H., (1995), '**C++ The Complete Reference Second Edition**', Osbourne McGraw-Hill.
12. Gonzalez, R.C., Woods, R.E., (1993), '**Digital Image Processing**', Addison-Wesley Publishing.
13. Dougherty, E.R., Giarina, C.R., (1987), '**Matrix Structured Image Processing**', Prentice-Hall.
14. Connolly, J.H., Edmonds, E.A., (Eds), (1994), '**CSCW and Artificial Intelligence**', Springer-Verlag.

## **6.2: SGI On-line Books**

- A. Myers, D., Deeth, E., (1994), '**IRIS Digital Media Tools Guide**', Silicon Graphics, Inc..
- B. Creek, P., Curtis, C., (1994), '**IRIS Digital Media Programming Guide**', Silicon Graphics, Inc..
- C. Bosder, C., (1994), '**Programming on Silicon Graphics: An Overview**', Silicon Graphics, Inc..
- D. Nye, A., (1992), '**Xlib Programming Guide**', O'Reilly & Associates, Inc..
- E. Graves, D., (1992), '**Impressario Programming Guide**', Silicon Graphics, Inc..
- F. Neider, J., Davis, T., Woo, M., (1995), '**OpenGL Programming Guide**', Addison Wesley.

## **6.3: World Wide Web Pages**

I. 'CSCW overview',  
<http://www.iao.fhg.de/Library/csw/OVERVIEW-en.html>

II. 'CVEs tutorial',  
<http://www.crg.cs.nott.ac.uk/~sdb/CVEs.html>  
This page included the following acknowledgement;

*'We would like to thank the following for their support in funding this research: EPSRC through the Virtuosi project and PhD studentships. The European Commission through the COMIC ESPRIT III Basic Research Action. We would also like to thank the following researchers who have contributed to this research: John Bowers, The University of Manchester; Lennart FahlÄn, The Swedish Institute of Computer Science; Tom Rodden and John Mariani, Lancaster University; and Research staff at British Telecom, Division, BICC, GPT, GEC Hirst Research Center, Nottinghamshire County Council and Nottingham Trent University for their contribution to the Virtuosi Project.'*

III. 'SGI-Format',  
<http://www.geo.unizh.ch/~heller/toporobot/Deutsch/Lieferanten/ToporobotFormate/SGI.html>

IV. 'Can anyone provide details on the SGI movie format?', SGI movie (FAQs)  
<http://rainbow.ldgo.columbia.edu/documentation/sgi-faq/movie/10.html>

V. 'Can anyone provide details of QuickTime movie format?', SGI movie (FAQs)  
<http://rainbow.ldgo.columbia.edu/documentation/sgi-faq/movie/11.html>