

Photo Based 3D Walkthrough

by

Lai Chan Yet

Dissertation submitted in partial fulfillment of
the requirements for the
Bachelor of Technology (Hons)
(Information Communication Technology)
NOVEMBER 2006

University Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

t

TK

5103.2

.Y48

2006

1) Three dimensional display system
2) IT BS -- Thesis

CERTIFICATION OF APPROVAL

PHOTO BASED 3D WALKTHROUGH

by

Lai Chan Yet

A project dissertation submitted to the
Information Communication Technology Programme
Universiti Teknologi PETRONAS
in partial fulfillment of the requirements for the
BACHELOR OF TECHNOLOGY (Hons)
(Information Communication Technology)

Approved by,



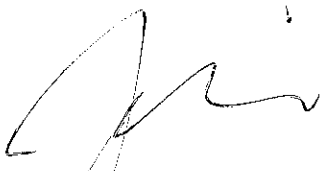
Mr. M. Nordin Zakaria

Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
NOVEMBER 2006

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



LAI CHAN YET

ABSTRACT

The objective of 'Photo Based 3D Walkthrough' is to understand how image-based rendering technology is used to create virtual environment and to develop a prototype system which is capable of providing real-time 3D walkthrough experience by solely using 2D images. Photo realism has always been an aim of computer graphics in virtual environment. Traditional graphics needs a great amount of works and time to construct a detailed 3D model and scene. Despite the tedious works in constructing the 3D models and scenes, a lot of efforts need to be put in to render the constructed 3D models and scenes to enhance the level of realism. Traditional geometry-based rendering systems fall short of simulating the visual realism of a complex environment and are unable to capture and store a sampled representation of a large environment with complex lighting and visibility effects. Thus, creating a virtual walkthrough of a complex real-world environment remains one of the most challenging problems in computer graphics. Due to the various disadvantages of the traditional graphics and geometry-based rendering systems, image-based rendering (IBR) has been introduced recently to overcome the above problems. In this project, a research will be carried out to create an IBR virtual walkthrough by using only OpenGL and C++ program without the use of any game engine or QuickTime VR function. Normal photographs (not panoramic photographs) are used as the source material in creating the virtual scene and keyboard is used as the main navigation tool in the virtual environment. The quality of the virtual walkthrough prototype constructed is good with just a little jerkiness.



ACKNOWLEDGEMENT

I am indebted and grateful to everyone who has provided both direct and indirect assistance to the completion of this project.

First and foremost, I would like to dedicate my gratitude to my loving family for their continuous support and encouragement. Mom, Dad, you are my superheroes, thanks for being there for me all the time.

I am greatly indebted to my project supervisor, Mr M. Nordin Zakaria, who had continuously monitored my progress during the course of the project and never shows a sign of uninvited for my frequent show up in front of his office door. Without his constructive comments, advices and guidance this project will not be a success.

I would also like to give credit to Wan Nursyamsiah who always tried to give a hand whenever I gave her a help sign in Yahoo messenger. She has been very supportive and I wish her all the best for her final year project. Thanks, for just letting me be a nerd.

Last but not least, I would also like to thank Mohd Sharul, Hiew Lu Cing, Jamie Tan Jia Nee, Aini Fadhilah, and Mohd Juzaili for their invaluable advice, comments and support.

LIST OF ABBREVIATION

IBR	Image Based Rendering
UTP	Universiti Teknologi Petronas
VR	Virtual Reality
SOI	Sea of Images
MPEG	Moving Picture Experts Group
DCT	Discrete Cosine Transformation
IDCT	Inverse Discrete Cosine Transformation
JPEG	Joint Photographic Expert Group
FDCT	Forward Discrete Cosine Transform
MB	Megabyte
EWMA	Exponentially Weighted Moving Average



TABLE OF CONTENTS

ABSTRACT	i
ACKNOWLEDGEMENT.....	ii
LIST OF ABBREVIATION.....	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES.....	iii
LIST OF TABLES	viii
CHAPTER 1 INTRODUCTION.....	1
1.1 Background study	1
1.2 Significant differences between Photo Based 3D Walkthrough and Panoramic Imaging.....	1
1.2.1 Process of taking the photographs.....	2
1.2.2 Real time walkthrough	3
1.2.3 Software used	3
1.3 Problem statement	3
1.3.1 Problem identification	3
1.3.2 Significant of the project	4
1.4 Objective and scope.....	5
1.4.1 Objectives	5
1.4.2 Scope of study	5
CHAPTER 2 LITERATURE REVIEW.....	7
2.1 Sea of Images	7
2.1.1 Compression	9
2.1.2 Data retrieval	10
2.2 MPEG	11
2.2.1 Sequences, pictures, and samples.....	11
2.2.2 Frames and fields.....	12
2.2.3 Pels and pixels	13
2.2.4 Compression vocabulary	14
2.2.5 MPEG compression.....	15
2.3 Discrete Cosine Transform.....	16

2.3.1 One-dimensional cosine transform	16
2.3.2 Two-dimensional cosine transform	19
2.4 Quantization	23
2.5 Prefetching mechanism	25
CHAPTER 3 METHODOLOGY	27
3.1 Methodology.....	27
3.1.1 Procedure identification.....	28
3.1.2 Preliminary data collection.....	28
3.1.3 Define problem statements and goals.....	29
3.1.4 Literature review and research	29
3.1.5 Design and construct prototype	31
3.1.6 Evaluation of the system	42
3.2 Tools Required	42
3.2.1 Digital Camera.....	42
3.2.2 Hardware	42
3.2.3 Software.....	43
CHAPTER 4 RESULTS AND DISCUSSION	44
4.1 Project prototype	44
4.1.1 Move forward and move backward	44
4.1.2 Move left and right to go to the next pathway.....	50
4.1.3 Turn around at a fixed spot to have a 180 degrees view at the surrounding .	55
4.2 Memory space consumption.....	62
4.3 Prefetching Mechanism	64
4.4 Prefetching and Caching Mechanism.....	68
4.5 Comparison between JPEG and BITMAP format.....	70
4.5.1 Image quality	70
4.5.2 Comparison between JPEG and BITMAP image file size.....	72
4.6 Speed (frame rate)	76
4.7 Smoothness of the virtual walkthrough.....	76
4.8 Quality of the virtual scene display	79
CHAPTER 5 CONCLUSION AND RECOMMENDATION	83
5.1 Conclusion.....	83
5.2 Future work	84
REFERENCES	87
APPENDICES	iix

LIST OF FIGURES

Figure 2.1 Multiresolution tree	10
Figure 2.2 Analog video frame and field structure.....	12
Figure 2.3 Component samples for MPEG-1 pels (1).....	13
Figure 2.4 Component samples for MPEG-1 pels (2).....	14
Figure 2.5 Even function	17
Figure 2.6 Odd function	17
Figure 2.7 Cosine function	18
Figure 2.8 Sine function	18
Figure 2.9 Discrete cosine transform.....	19
Figure 2.10 8x8 block.....	20
Figure 2.11 2D DCT.....	20
Figure 2.12 2D DCT formula	21
Figure 2.13 2D DCT 8x8 block data	21
Figure 2.14 Zigzag scan.....	22
Figure 2.15 2D IDCT formula.....	22
Figure 2.16 Prediction of moving direction using (a) mean, (b) window, (c) EWMA ..	25
Figure 3.1 Project development steps.....	28
Figure 3.2 Virtual walkthrough architecture design.....	31
Figure 3.3 Steps to take panoramic photographs.....	33
Figure 3.4 Capture images 180° around a single point of rotation	34
Figure 3.5 Photographs taken 180° around the single point of rotation with each photographs overlap each other by more then 50 % horizontally	35
Figure 3.6 Camera orientation.....	36
Figure 3.7 Usage of multidimensional array	41
Figure 4.1 Moving forward and backward (1)	45
Figure 4.2 Moving forward and backward (2)	46
Figure 4.3 Moving forward and backward (3)	47
Figure 4.4 Alert message (1)	48
Figure 4.5 Alert message (2)	49
Figure 4.6 Move left and right (1)	50
Figure 4.7 Move left and right (2)	51
Figure 4.8 Move left and right (3)	52
Figure 4.9 Alert message (3)	53
Figure 4.10 Alert message (4)	54
Figure 4.11 180 degrees view (1).....	56
Figure 4.12 180 degrees view (2).....	57
Figure 4.13 180 degrees view (3).....	58

Figure 4.14 180 degrees view (4)	59
Figure 4.15 180 degrees view (5)	60
Figure 4.16 180 degrees view (6)	61
Figure 4.17 Total number of photographs needed with the corresponding number of footsteps.....	62
Figure 4.18 Memory consumption in MB with the corresponding number of footsteps	63
Figure 4.19 Mean Prediction Scheme (1)	65
Figure 4.20 Mean Prediction Scheme (2)	66
Figure 4.21 Mean Prediction Scheme (3)	67
Figure 4.22 Caching (1).....	68
Figure 4.23 Caching (2).....	69
Figure 4.24 JPEG format photograph.....	71
Figure 4.25 BITMAP format photograph.....	72
Figure 4.26 Total image file size needed by JPEG and BITMAP format file.....	74
Figure 4.27 JPEG and BITMAP file size consumed with corresponding number of footsteps in the walkthrough	75
Figure 4.28 Difference in angle when taking photographs	77
Figure 4.29 Changes in viewing angles during the walkthrough.....	78
Figure 4.30 Original JPEG image before conversion.....	81
Figure 4.31 BITMAP image after conversion	82
Figure 5.1 Client server architecture	84



LIST OF TABLES

Table 2.1: Default luminance quantization table for intra coding.....	24
Table 4.1 Image format and image quality.....	70
Table 4.2 Image format and image file size	73
Table 4.3 Image qualities and file size before and after conversion	80

CHAPTER 1

INTRODUCTION

1.1 Background study

Image based rendering (IBR) used photographs as a source to create three dimensional photorealistic objects and complete scenes. Photo realism has always been an aim of computer graphics in virtual environment. IBR technique has provided several advantages over traditional graphics pipelines. There are many different IBR techniques and approaches have been used to create the virtual environment for virtual walkthrough and all of them used images as the source materials. Different approaches and different techniques will produce different virtual environment, the techniques and approaches used are greatly depends on the nature and usage of the application produced. Some applications used the two dimensional images to transform into three dimensional models in order to create the three dimensional virtual environment, where else certain applications only used two dimensional images to create a two dimensional virtual environment. Thus with IBR, photorealistic system can be produced with less computing power due to the lack of complex models. The introduction of IBR has also shorten the development time of the traditional computer graphic system which is always view as never ending project due to the long delivery timeline.

1.2 Significant differences between Photo Based 3D Walkthrough and Panoramic Imaging

Many people confused Photo Based 3D Walkthrough with Panoramic Imaging due to both use sequences of photographs in generating the view of the surrounding

environment. However, the only similarity between this two is that both used photographs as source material to create the walkthrough. Below discussed the main differences between Photo Based 3D Walkthrough and Panoramic Imaging in detail:

1.2.1 Process of taking the photographs

The major difference between Photo Based 3D Walkthrough and Panoramic Imaging is the process of how the photographs are taken. In Photo Based 3D Walkthrough, no specific measurements or rules need to be followed in capturing the panoramic photographs, where else, in Panoramic Imaging, there are rules, specific measurement and angles need to be followed. The process of taking panoramic photographs for Panoramic Imaging is much more tedious compare to Photo Based 3D Walkthrough; measurement must be made before each panoramic photograph is taken. If the rules are not followed or there is a slight mistake in the measurement during the process of taking the panoramic photographs, there will be problems in creating the Panoramic Imaging.

In Panoramic Imaging, the amount of tedious work does not just stop after the large amount of panoramic photographs has been taken, the photographs must be stitched using Stitching software. The process of stitching all the photographs is tedious and time consuming too. If there are any errors in taking the panoramic photographs, problems might occur during the stitching process because the panoramic photographs will not be able to stitch together smoothly. Correction must be taken before the panoramic photographs can be stitched.

As a result, Photo Based 3DWalkthrough is an improved method to help the developer to implement more realistic and real time 3D virtual walkthrough in an easier, more efficient and productive way.

1.2.2 Real time walkthrough

Photo Based 3D Walkthrough is real time walkthrough where else Panoramic Imaging is not real time walkthrough. In Photo Based 3D Walkthrough, the viewer can navigate or walk in the virtual environment just like how a person can move around in a 3D game environment. In Panoramic Imaging, a viewer cannot experience real time walkthrough; the viewer only can observe the 360 degrees view of the surrounding from a fixed spot. The most a person can navigate or move around is through the hot spot available in the Panoramic Imaging. However by navigating using the hot spot available to move to another place, the viewer is jumping from one place to another rather than walking in the virtual environment and enjoy the change of scenery when every foot step is taken.

1.2.3 Software used

The software used in creating the Panoramic Imaging is Quick Time VR or Flash, which is customized and created for the use of implementing 360 degrees panoramic imaging. This made Panoramic Imaging not flexible because the developer has no control over the source codes. On the other hand, Photo Based 3D Walkthrough is written in OpenGL and C++ programming language in the Microsoft .Net framework. The developer is able to modify the program and add in new features to enhance the program anytime.

1.3 Problem statement

1.3.1 Problem identification

Traditional graphics needs a great amount of works and time to construct a detailed 3D model and scene. Despite the tedious works in constructing the 3D models and scenes, a lot of efforts need to be put in to render the constructed 3D models and

scenes to enhance the level of realism. Traditional geometry-based rendering systems fall short of simulating the visual realism of a complex environment and are unable to capture and store a sampled representation of a large environment with complex lighting and visibility effects. Many industries such as Architect Company, House Development Company, and Power Plant have started to adopt technology such as virtual walkthrough into their daily business function. Fast delivery with the desired quality of virtual environment for the virtual walkthrough is very crucial in the world of competing business. The great amount of unnecessary efforts and time wasted in creating an unrealistic virtual walkthrough through the traditional graphics method delays the delivery of the virtual walkthrough product. Thus, creating a virtual walkthrough of a complex real-world environment remains one of the most challenging problems in computer graphics.

1.3.2 Significant of the project

The goal of this project is to create a smooth and photo realism IBR walkthrough for complex real world scenes based on photographs and with zero 3D model creation and rendering works.

The advantages brought by the above approach are:

- i. Tedious modeling work by using 3D creation software can be replaced by less work and more realistic images, which reduce the unnecessary manual work and render complexity.
- ii. More realistic and better quality walkthrough environment with higher degree of realism can be achieved.
- iii. Shorter development time and faster delivery of the virtual walkthrough product.
- iv. Provide better understanding on the latest technology in Virtual Reality development.

- v. To ignite and wide spread the development of Virtual Reality product by using IBR technology.
- vi. Photorealistic system can be produced and run on less computing power computer due to the lack of complex models.

Possibly, IBR technology may replace the tedious traditional graphics modeling in the future to ease the creation of complex virtual scene for a virtual walkthrough.

1.4 Objective and scope

1.4.1 Objectives

- i. To understand how image-based rendering technology is used to create virtual environment and virtual walkthrough.
- ii. To develop a prototype which is capable of providing a smooth real-time virtual walkthrough experience in the selected area in UTP campus.

1.4.2 Scope of study

Study focuses on constructing a smooth virtual walkthrough for a selected area in the VR lab by using OpenGL with C++ and the captured photographs of the VR lab only. Virtual walkthrough with multiple directions is created. However, to fit the project time frame, the virtual walkthrough is created only for a small selected area as an experimental purpose for the research of this new method.

A virtual walkthrough prototype will be designed and implemented. Emphasis is on the creation of a virtual walkthrough system solely based on images using OpenGL with C++. The prototype will be able to store the captured images, retrieve the



stored images to construct a virtual walkthrough. The viewer will be able to move forward, backward and turn around in 180 degrees angle in the developed prototype.

CHAPTER 2

LITERATURE REVIEW

2.1 Sea of Images

Aliaga, Funkhouser, Yanovsky, and Carlbom stated that the most challenging problem in computer graphic is to create an interactive walkthrough of a complex real world environment (p. 331). Currently there is no interactive system which can regenerate the photorealistic real world environment. According to Aliaga, Funkhouser, Yanovsky, and Carlbom, image-based rendering achieve photorealism by using a set of photographs to render novel viewpoints without the need of constructing detail 3D model. However, the current IBR technique is only capable of generating IBR walkthrough for a small scene or environment with low geometric complexity (p. 331). The 'Sea of Images' is a method proposed by Aliaga, Funkhouser, Yanovsky, and Carlbom to create an IBR walkthrough system that supports an interactive experience for large and complex real world scene. In their article, Aliaga, Funkhouser, Yanovsky, and Carlbom explained that the 'Sea of Images stands for a collection of images every couple inches throughout a large environment (p. 331). To generate the 'Sea of Images', Aliaga, Funkhouser, Yanovsky, and Carlbom captured omnidirectional images on an eye height plane by moving a video camera mounted on a motorize cart back and forth in a zigzag pattern throughout the environment. Aliaga, Funkhouser, Yanovsky, and Carlbom explained that the captured images are then compressed in a multiresolution hierarchy to enable them to be access efficiently. Time critical algorithm is used to prefetch the relevant images and feature based morphing method is used to reconstruct images during the interactive walkthrough (p. 331).

The difficult computer vision problems such as 3D construction and surface reflectance modeling can now be replaced with easier problem from SOI approach, which are motorized cart navigation, data compression and working set management. There are four advantages provided by the SOI approach. As stated by Aliaga, Funkhouser, Yanovsky, and Carlbom, first, it enables precise image reconstruction for novel views in environments with specular surfaces, a great amount of geometrical detail, and complex visibility changes. Second, it does not require accurate geometric model to produce novel images over a wide range of viewpoints. Third, it provides a method for images-based modeling of a large environment without sophisticated gaze planning. And lastly, it supports of rendering inside looking out images in an IBR interactive walkthrough system (p. 332).

The three main challenges in implementing an IBR interactive system by using the ‘Sea of Images’ are obtaining a large set of images over a large area in a practical manner, compressing the massive amount of captured images, and accessing a large out of core data set for real time walkthrough. The authors of this paper have proposed and experiment the below solutions for each of the problems mentioned above in their research work. According to Aliaga, Funkhouser, Yanovsky, and Carlbom, an omnidirectional camera on a motorized cart is built from off the shelf components to capture a dense of images on an eye height plane. The motorized cart moved the omnidirectional camera in a zigzag pattern through a large environment. This system is small enough to move around anywhere a person can walk through and more pictures are captured for areas where viewer may want to study object in detail. Compression is achieved by implementing a multiresolution IBR representation to exploit coherence in nearby images. While predictive fetching and caching algorithm are used to load images from disk as a viewer moves through an environment interactively (p. 333).

2.1.1 Compression

The captured data sets contain thousand of images, requiring gigabytes of storage. According to Aliaga, Funkhouser, Yanovsky, and Carlbom, among the thousand of images, only a small portion will be used to reconstruct an image for a novel viewpoint and there is great coherence among images from adjacent viewpoints (p. 334). An assumption is made by the author that there is enough disk storage to hold the entire data set, the only problem is to compress the data into a form that it can be access efficiently during an interactive walkthrough. Aliaga, Funkhouser, Yanovsky, and Carlbom proposed to take advantage of the data redundancy while optimizing the data layout for cache coherence. There are two options to achieve this, the first option is to compress every captured data independently in JPEG file (this is done automatically by the digital camera during capture), however this method does not take advantage of inter viewpoint redundancy, which can improve both storage and bandwidth utilization. The second option is to use prediction and replace some images with the residual or difference between the original image and the predicted image (p. 334). Compression can be gained because the residual has less energy than the original image. However the difficult part is to find a proper I-frames spacing. There is a conflict between using frequent I-frames or infrequent I-frames. Very little compression can be gained from frequent I-frames where else infrequent I-frames yield more compression but make non linear images access inefficient and poor cache utilization for walkthrough application. By weighing the pros and cons, the authors optimize both compression and cache utilization by storing images in a multiresolution hierarchy. The strategy proposed by Aliaga, Funkhouser, Yanovsky, and Carlbom is to use a nested tree structure where the root of each tree is an I-frame and the rest of the nodes are the P-frames. Only P-frames must be read for nearby viewpoints lower in the hierarchy because the system caches image data associated with interior nodes in the hierarchy. Thus improve the disk to memory bandwidth utilization (p. 334).

According to Aliaga, Funkhouser, Yanovsky, and Carlbom, the multiresolution data structure is a binary tree built bottom up by using half edge collapse operation. A set of nodes representing each captured image is created to initialize the tree. The priority for every edge is computed by using triangulation of the nodes. The highest priority edge is collapsed and replaces the node at the edge end point with a new common parent node, as shown in Figure 2.1.

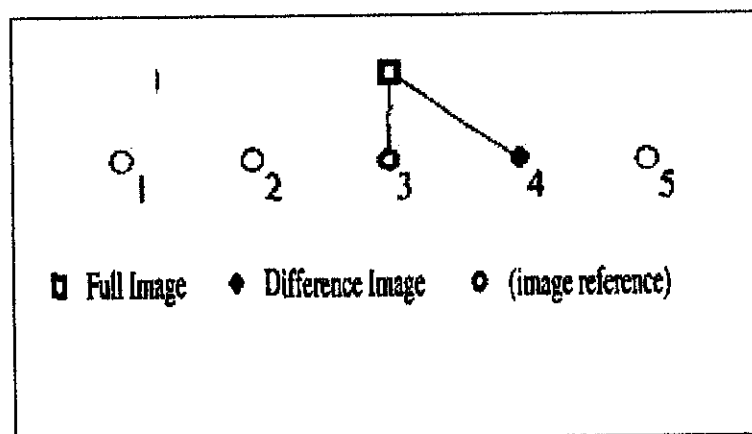


Figure 2.1 Multiresolution tree

The parent node is placed at the same viewpoint location as one of its children and the original image is replaced with an image referenced to prevent introducing resampled images in the hierarchy. Edges between the most similar images are collapsed first. A faster approximation is to collapse the shortest edge (p. 334).

2.1.2 Data retrieval

Aliaga, Funkhouser, Yanovsky and Carlbom state during the virtual walkthrough, the stored images need to be pre-load and cache from the disk for rendering novel view as a viewer navigates interactively. The task is to develop a time-critical algorithm to load and cache data in a manner that produce the highest quality images while maintaining interactive frame rate (p.335). The algorithm used must

be able to ensure that a set of images is always surrounding the viewpoints so that a good reconstruction and smooth transition from a set of images to another can be achieved. This algorithm should also exhibit graceful degradation and return maximum quality when failure occurs. Aliaga, Funkhouser, Yanovsky and Carlbom propose an asynchronous prefetching process to maintain a cache of images in main memory. So that based on the viewer's velocity, the images which are needed next can be determined (p.335). The prefetcher maintains two linked lists of tree nodes, the evict list and the fetch list. The evict list defined a cut through the tree such that all nodes above this cut are in cache. The fetch list contain all the children of the evict list nodes. These two lists are sorted by priority. The first image in the fetch list has the highest probability of being needed next, and the last image in the evict list has the highest probability of being needed next. The authors further described that to display each frame at run time; the renderer uses the current cut to determine which images to use for reconstructing the current view. To update the cut, swap nodes between the two lists. Use observer latest viewpoint and predicted velocity to compute for all nodes in the evict and fetch list the probability that its image is needed next (p.335). When the observer gets closer to the nodes, the probability of the node increased.

2.2 MPEG

2.2.1 Sequences, pictures, and samples

According to Mitchell, Pennebaker, Fogg, and LeGall (2001), MPEG video sequence is made up of many single pictures that occur at fix time increment. Each of these color pictures has three components which are luminance and two chrominance (p. 4). Luminance component gives us a monochrome picture (grayscale), while chrominance defined the color hue and saturation of the picture.

Mitchell, Pennebaker, Fogg, and LeGall (2001) stated that these three components are not expressed in the common RGB because they are mathematically equivalent color representation that can be compressed more efficiently. Each component is made up of a two dimensional grid that consists of array of samples. Each horizontal line of samples is called raster line, or in other words a raster line consists of many samples. The intensity of the component at any particular point on the raster line is digitally represented by the samples (p. 4). The chrominance component is sampled at a lower spatial resolution compare to the luminance component due to the insensitivity of human eyes in resolving rapid spatial changes in chrominance.

2.2.2 Frames and fields

Mitchell, Pennebaker, Fogg, and LeGall (2001) mentioned that broadcast analog video sequences are temporally subdivided into frames and raster lines and each frame is divided into two interlaced fields, as shown in Figure 2.2.

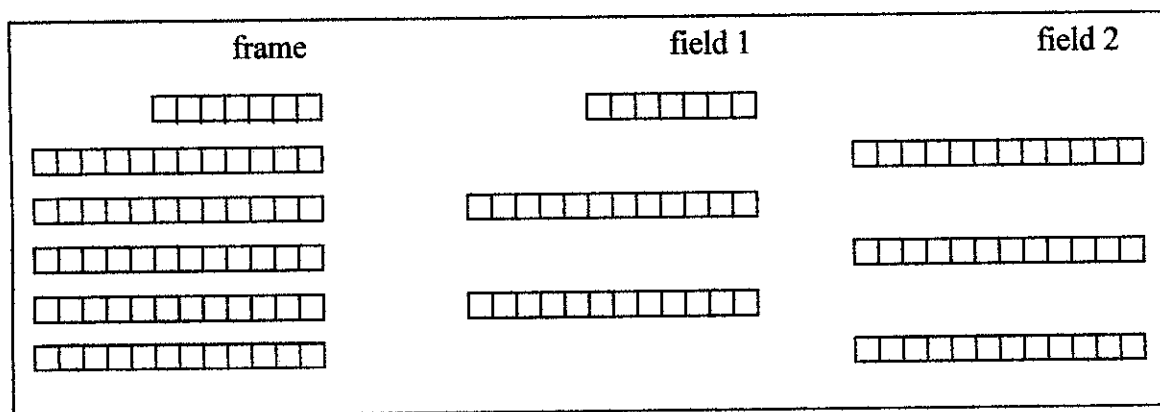


Figure 2.2 Analog video frame and field structure

The relationship between analog video frames and fields and MPEG-1 pictures is undefined. However MPEG-2 allows interlaced pictures, coding them either as individual fields or full frames (p. 5). The artifact due to blanking is usually cropped

before the picture is presented to an MPEG encoder. Thus the MPEG decoder must be intelligent enough to recreate the cropped edge areas.

2.2.3 Pels and pixels

A pel is formed by the component samples values at a particular point in a picture. A pel will have three samples if all three components are using the same sampling grid (Mitchell, Pennebaker, Fogg, and LeGall, 2001, p. 6). Since the eye is insensitive to rapid spatial variation in chrominance information, MPEG subsamples the chrominance component by using a chrominance grid that is a factor of 2 lower in resolution compare to luminance grid. Therefore, a pel is defined to be the highest sampling resolution, but not all samples in the pel are having the same resolution. Figure 2.3 and 2.4 shows a sketch of an MPEG pels.

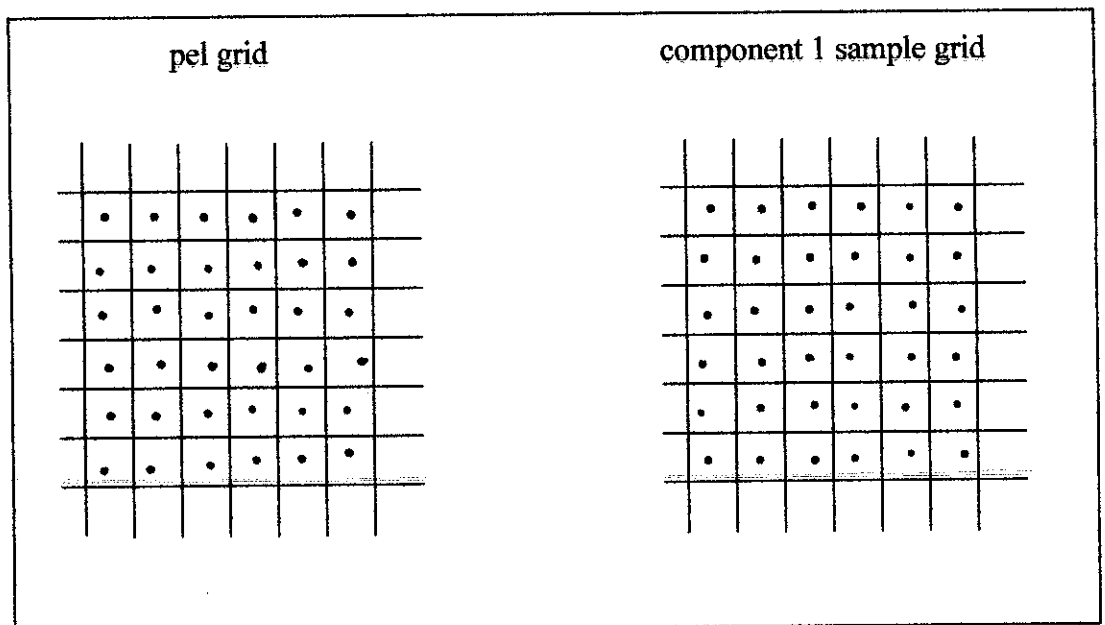


Figure 2.3 Component samples for MPEG-1 pels (1)

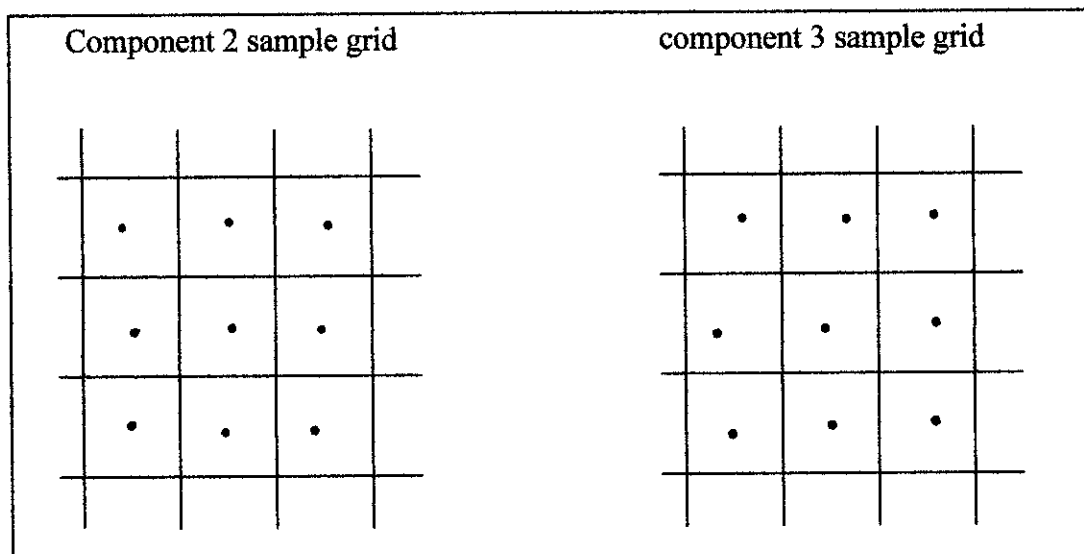


Figure 2.4 Component samples for MPEG-1 pels (2)

According to Mitchell, Pennebaker, Fogg, and LeGall (2001), the positioning of the lower resolution samples relative to the highest resolution samples must be defined when a lower sampling resolution is used for some components in a pel (p. 7).

2.2.4 Compression vocabulary

Mitchell, Pennebaker, Fogg, and LeGall (2001) stated that the compression techniques used in MPEG can be classified as intra and non intra (p. 7). The difference between these two techniques is that whether a picture is compressed by using the information solely from that particular picture or information from one or two other pictures displaced in time. As denoted by the name, intra technique only used information within the picture for compression whereby non intra technique used information from other pictures for compression. According to Mitchell, Pennebaker, Fogg, and LeGall (2001), the input into data compressor is called data source while the output from decompressor is called reconstructed data. The reconstructed data can be lossless or lossy depending on the compression technique

applied (p. 7). If the reconstructed data is lossless, this means the reconstructed data and the source data is exactly matching. Where else, if the reconstructed data is lossy, this shows that the reconstructed data is only a good approximation of the source data. Lossless compression technique is far less efficient than lossy compression technique which is why most data compression algorithm combined both lossy and lossless elements. According to Mitchell, Pennebaker, Fogg, and LeGall (2001), for a compression to be effective, the lossy elements should be able to selectively ignore structure in the data the human eye cannot see (p. 7).

2.2.5 MPEG compression

As mentioned by Mitchell, Pennebaker, Fogg, and LeGall (2001) the key aspects of moving picture compression is the similarity between pictures in a sequence. Smaller differences between pictures will lead to greater compression (p. 11). This is because when there is no difference between pictures, the sequence of pictures can be just simply coded as a single picture followed by a few bits to tell the decoder to repeat the picture. Mitchell, Pennebaker, Fogg, and LeGall (2001) explained that a technique which can code the first picture without reference to neighbor picture is needed to code the first picture. However this technique will be used at regular intervals to occasionally code the picture in a sequence independent of the neighboring pictures (p. 11). The pictures in a sequence needed to be coded independently sometimes because bits errors during transmission may make image reconstruction from the reference image difficult. Thus the sequence reconstructed by the decoder may build up to unacceptable levels. Mitchell, Pennebaker, Fogg, and LeGall (2001) explained that the high compression of MPEG is achieved by coding most of the pictures as differences relative to neighboring pictures. These can be accomplished in a few ways such as copying the parts of the picture where no significant changes occur and use motion compensation technique to predict the parts of the image which are displaced due to motion (p. 11).

2.3 Discrete Cosine Transform

As mentioned by Mitchell, Pennebaker, Fogg, and LeGall (2001), DCT has properties that can simplify coding models and make the coding more efficient (p. 25). DCT is a method that decomposes the block of data into a weighted sum of spatial frequencies. Mitchell, Pennebaker, Fogg, and LeGall (2001) explained that each of the spatial frequency patterns has a corresponding coefficient; these coefficients are the amplitude needed to represent the spatial frequency patterns in the block of analyzed data (p. 25). The 8 x 8 blocks of chrominance samples that represent colors in terms of the presence of red and blue are the unit processed by the DCT. As mentioned above, a lower resolution is used for chrominance due to human eyes capability to resolve high frequencies luminance better than chrominance.

2.3.1 One-dimensional cosine transform

Mitchell, Pennebaker, Fogg, and LeGall (2001) mentioned in their book that the complex variation in video signal amplitude can be expressed as a sum of simple sine or cosine waveforms. In order for the sum of sine or cosine waveforms to exactly matched the signal variation, the sum of sine or cosine waveforms must have the right spatial frequencies and amplitudes. A given function, $f(x)$ can be either having odd or even symmetry. An even function is the one that obeys the $f(x) = f(-x)$ relationship, while an odd function is the one that obeys the $f(x) = -f(-x)$ relationship (p. 34). Please refer to Figure 2.5 and Figure 2.6. As shown in Figure 2.5, it is clear that an even function has even symmetry across the vertical axis while in Figure 2.6, it show that odd function must be negated when reflected across this axis. Figure 2.7 shows a cosine function. A cosine function is an even function because it has mirror symmetry across the vertical axis. The sine function in Figure 2.8 has an odd symmetry. As a conclusion, cosine transform has even symmetry while sine transform has odd symmetry.

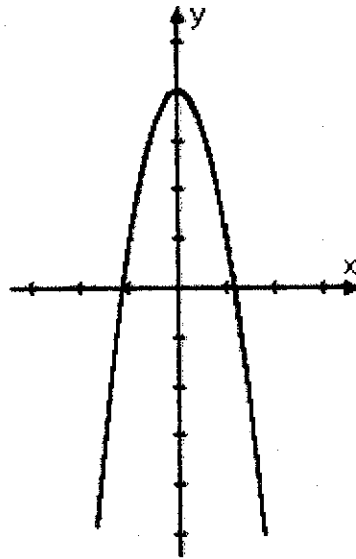


Figure 2.5 Even function

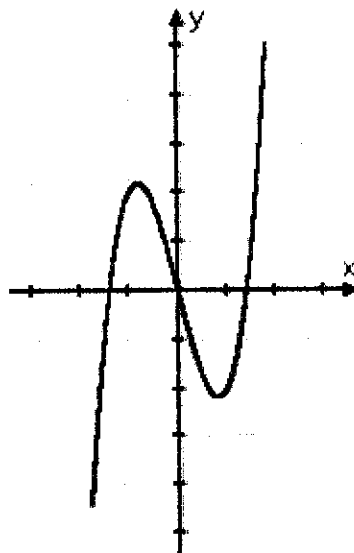


Figure 2.6 Odd function

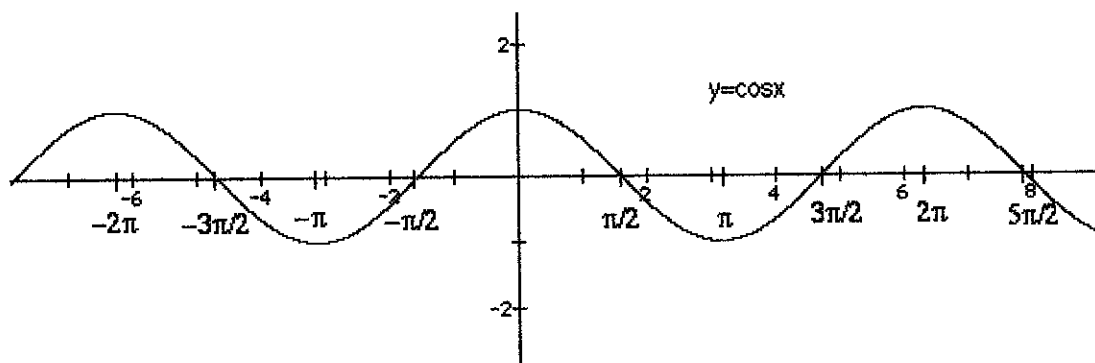


Figure 2.7 Cosine function

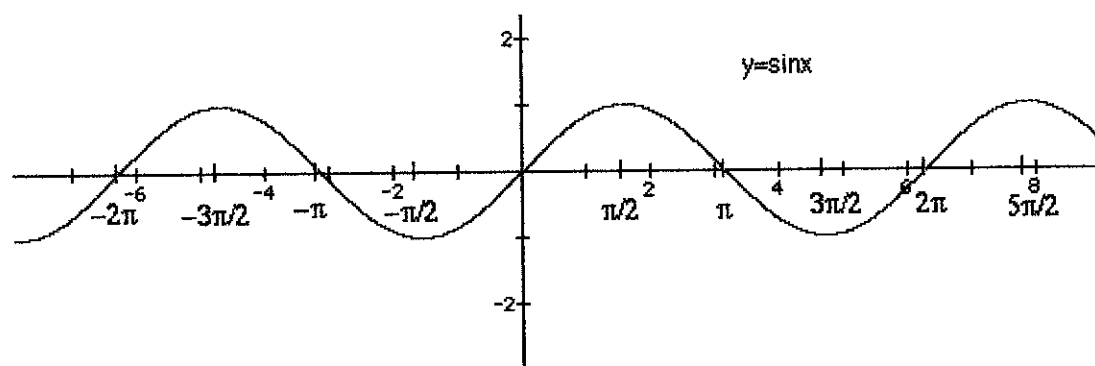


Figure 2.8 Sine function

According to Mitchell, Pennebaker, Fogg, and LeGall (2001), digital data systems take samples of the data at discrete intervals, the discrete intervals are set by the number of samples needed for a given picture resolution. When the smoothly varying cosine function in Figure 2.7 is sampled at same interval as shown in Figure 2.9, it becomes discrete cosine transform.

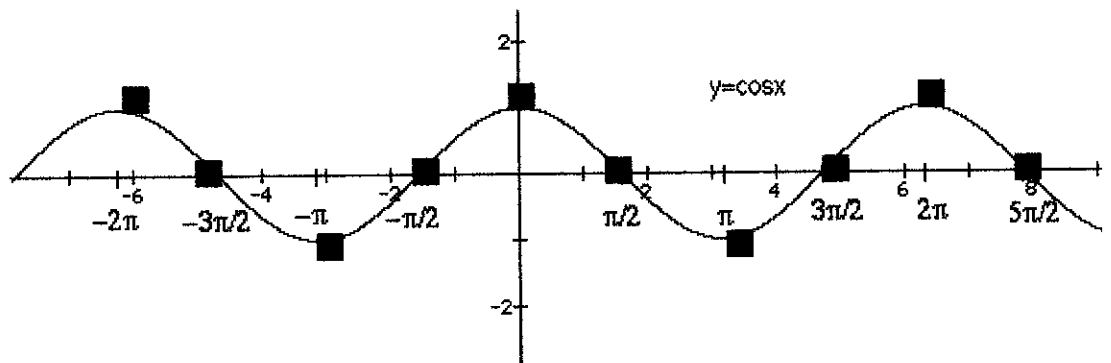


Figure 2.9 Discrete cosine transform

The upper bound frequencies that can be reproduced in a sampled system are given by the Nyquist limit of two samples per cycle. Frequency above the Nyquist limit will appear as lower frequency oscillations; this effect is called aliasing. Aliasing is not desirable because it is usually suppressed by low pass filtering before sampling it (p. 37).

2.3.2 Two-dimensional cosine transform

Mitchell, Pennebaker, Fogg, and LeGall (2001) explained in their book that MPEG uses 2D 8x8 form of DCT to compress pictures (p. 42). A picture is divided into multiple 8x8 blocks of data, and the value of the brightness of each pixel in the block is represented in the scale of 0 to 255, as shown in Figure 2.10.

120	108	90	75	69	73	82	89
127	115	97	81	75	79	88	95
134	122	105	89	83	87	96	103
137	125	107	92	86	90	99	106
131	119	101	86	80	83	93	100
117	105	87	72	65	69	78	85
100	88	70	55	49	53	62	69
89	77	59	44	38	42	51	58

Figure 2.10 8x8 block

According to Mitchell, Pennebaker, Fogg, and LeGall (2001), the 2D DCT is composed of separate horizontal and vertical 1D DCT, as shown in Figure 2.11.

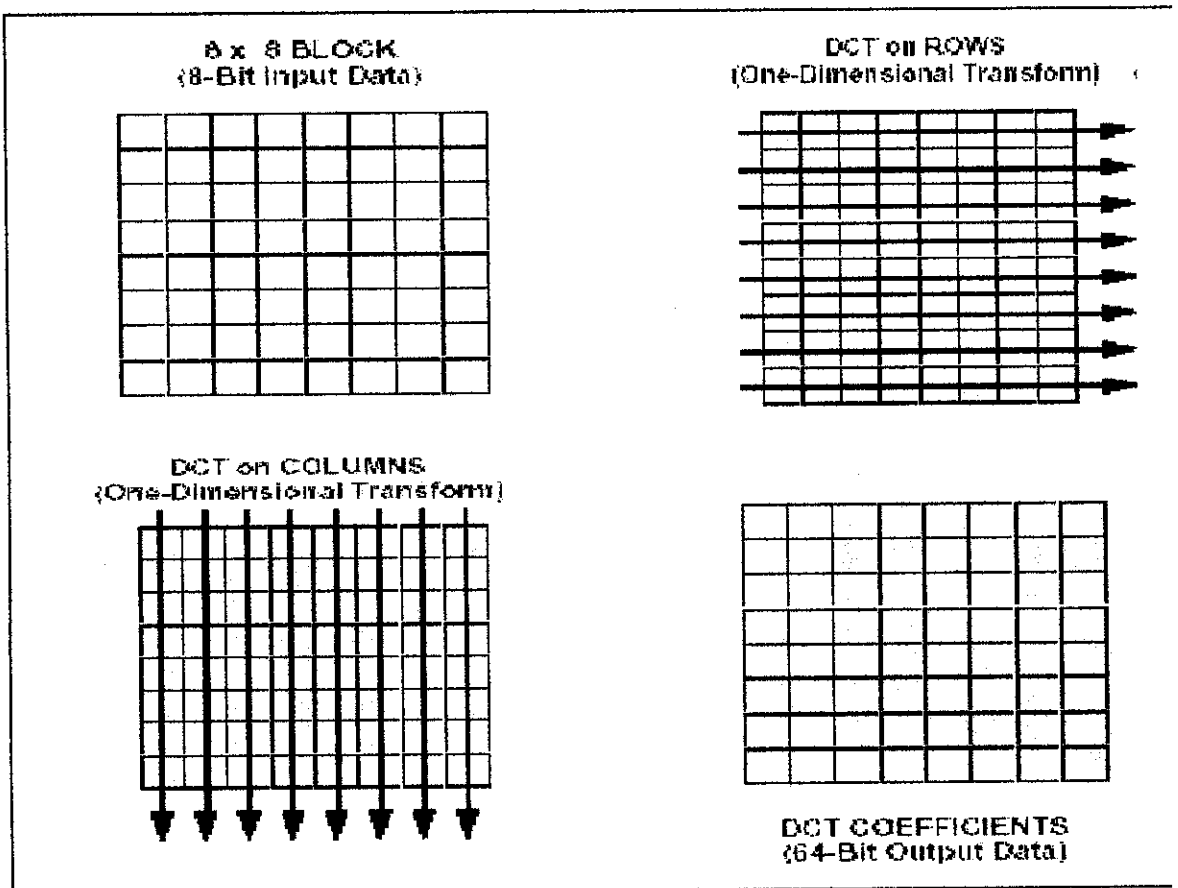


Figure 2.11 2D DCT

If a 2D array of samples, $f(x,y)$, need to be transformed into a 2D DCT, first, run the horizontal 1D DCT to transform each row of array into 1D DCT. Second, run the vertical 1D DCT. The order in which the horizontal and vertical transform are taken does not affect the result, since they are separate process (p. 43). The following discrete cosine transform formula in Figure 2.12 is used to transform the above 8x8 value in Figure 2.10 into 2D DCT. Where $f(x,y)$ is the brightness of the pixel at position $[x,y]$. The end result is also an 8x8 array, which is shown in Figure 2.13.

$$F(u, v) = \frac{C_u}{2} \frac{C_v}{2} \sum_{y=0}^7 \sum_{x=0}^7 f(x, y) \cos \left[\frac{(2x+1)u\pi}{16} \right] \cos \left[\frac{(2y+1)v\pi}{16} \right]$$

with:

$$C_u = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0, \\ 1 & \text{if } u > 0 \end{cases}; C_v = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } v = 0, \\ 1 & \text{if } v > 0 \end{cases}$$

Figure 2.12 2D DCT formula

```

700 90 100 0 0 0 0 0
 90  0  0 0 0 0 0 0
-89  0  0 0 0 0 0 0
  0  0  0 0 0 0 0 0
  0  0  0 0 0 0 0 0
  0  0  0 0 0 0 0 0
  0  0  0 0 0 0 0 0
  0  0  0 0 0 0 0 0
    
```

Figure 2.13 2D DCT 8x8 block data

decorrelation since the pictures in sequence for non intra coding are already fairly well decorrelated, it still helps in compressing the data rather than none at all (p. 28). However, practically DCT works well for inter and intra coding.

2.4 Quantization

According to Mitchell, Pennebaker, Fogg, and LeGall (2001), the process of quantization is to represent the high spatial frequencies coefficients with low precision. Basically quantization is to reduce the precision of the DCT coefficient. Quantization is done by dividing the DCT coefficients by the quantization value (a nonzero positive integer), the result is usually in integer and fraction, and thus the quotients need to be rounded to the nearest integer according to the MPEG rules (p. 27). Mitchell, Pennebaker, Fogg, and LeGall (2001) stated that the quantized DCT values are the one that are transmitted to the decoder (p. 46). Bigger quantization value is preferred even though it offers lower precision because lower precision coefficients can be transmitted to the decoder in fewer bits. Mitchell, Pennebaker, Fogg, and LeGall (2001) also stated that large quantization value allows the encoder to selectively discard high spatial frequency activity that human eyes cannot perceive (p. 27). According to the authors, the quantizing values are chosen according to the principles based on human visual system in order to minimize perceived distortion in the reconstructed pictures (p. 46).

There are two quantization table in MPEG-1, one for intra coding and one for non intra coding (Mitchell, Pennebaker, Fogg, and LeGall, 2001, p.91). The default intra coding table is as shown in Table 2.1. It has a distribution of quantizing value that is roughly in accord with the frequency response of the human eye.

Table 2.1: Default luminance quantization table for intra coding

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

The non intra default MPEG quantization table is a flat with a fixed value of 16 for all coefficients. However the proper choice of a quantization value for a given function should be set according to the spatial frequency of the DCT function and the response of human visual system to that frequency.

As a conclusion quantization is a very powerful technique to control and reduce bit rate even when DCT does not improve decorrelation much.

2.5 Prefetching mechanism

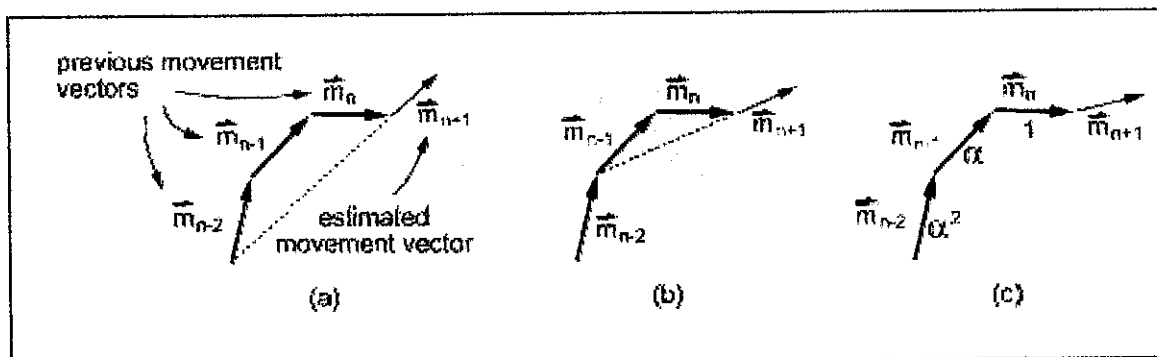


Figure 2.16 Prediction of moving direction using (a) mean, (b) window, (c) EWMA

According to Chim, Lau, Leong and Si (2003), to enable prefetching, the client must be able to maintain a viewing profile of the viewer; a viewing profile is a list of historical movement vectors, $\{ \vec{m}_{n-1}, \vec{m}_n, \dots \}$. The moving direction and location of a viewer at a particular time is used to calculate each vector. The n th movement vector, \vec{m}_n is determined when a client move to a new location loc_n with a new orientation. The $(n+1)$ th movement vector is predicted as \vec{m}_{n+1} by the client and request for the objects that will be needed when the viewer is at location loc_{n+1} , this can save future request since the object is already cached (p. 508). Chim, Lau, Leong and Si (2003) explained that there are three types of prediction schemes, which are mean, window and EWMA as shown in Figure 2.16. Mean scheme predicts the next movement vector \vec{m}_{n+1} as the average of the previous n movement vectors as shown in Figure 2.16 (a). On the other hand, in window scheme, each viewer has a window of size W which holds the previous W movement vectors. The average of the W most recent vectors is used to predict the next movement vector, please refer to Figure 2.16 (b). However, the flaw in this method is that all movement vectors within the window have equal effect on the prediction. The third prediction scheme is EWMA, this method provides better real movement

approximation and it can adapt to changes in viewers' moving patterns quickly. Exponential decreasing weight is assigned to each previous movement vector, \vec{m}_i ; with the most recent movement vector having the weight of 1, the previous having the weight of α , and the next previous having the weight of α^2 . Please refer to Figure 2.16 (c) for better understanding. The formula $\vec{m}_{\alpha+1} = \alpha \vec{m}_n + (1 - \alpha) \vec{m}_n$ is used to predict the new (n+1)th movement vector (p. 508).

CHAPTER 3

METHODOLOGY

3.1 Methodology

The methodology used in implementing this project is largely based on the method called 'Sea of Images' proposed by Aliaga, Funkhouser, Yanovsky, and Carlbom. The three main procedures proposed by Aliaga, Funkhouser, Yanovsky, and Carlbom in creating an IBR virtual walkthrough is to *take photos, store the photos taken and retrieve the correct photos during the virtual walkthrough to construct the virtual scene.* Below explained each of the steps taken in making this project a success:

3.1.1 Procedure identification

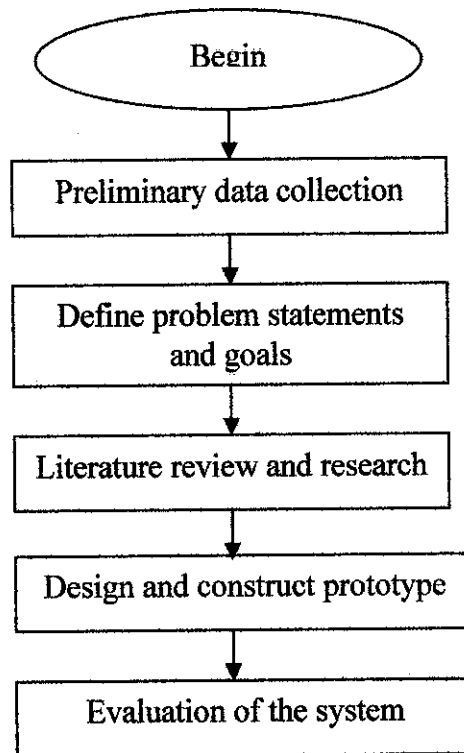


Figure 3.1 Project development steps

3.1.2 Preliminary data collection

Research work has been carried out on the relevant work done thus far as a guide in understanding the fundamental concepts of using IBR method to create a virtual walkthrough. Articles and journals related to image based rendering, virtual reality walkthrough, taking and stitching panoramic images has been read to grasp a basic idea on each of the steps involved in creating image based rendering virtual reality walkthrough.

3.1.3 Define problem statements and goals

Through the information get from the articles read as well as further discussion and consultation on the topics related to virtual walkthrough with Mr. M. Nordin Zakaria, it is understood that it is difficult and time consuming to reproduce the visual experience of walking through a large photorealistic environment by using traditional graphics method. Creating a realistic virtual walkthrough through traditional computer graphics approach will involve creating the detail 3D models and applying the appropriate or matching texture for each of the 3D model created to enhance the level of realism. However the amount of tedious work does not just stop here, the lighting effect as well as the smoothness of the 3D model surface need to be taken care of.

The objective of this project is to carry out a research on IBR algorithms and techniques to find out how to use IBR approach to implement a realistic virtual walkthrough to overcome the problem of the traditional geometry-based rendering system.

3.1.4 Literature review and research

Necessary research has been done to find out how to take a dense of panoramic images which are needed to create the virtual walkthrough. Lots of articles and journals on photo based 3D walkthrough are reviewed to use as a guide in taking the photographs and creating the 3D walkthrough. Other than that, journals and articles regarding on IBR and SOI have been reviewed to understand more on the design and architecture of the walkthrough system. The three journals used as the major references in this project are Sea of Images (2001) by Aliaga, D. G., Funkhouser, T., Yanovsky, D. and Carlbom, I., Image-based rendering for real-time applications by Bauer, M and CyberWalk: A Web-Based Distributed Virtual Walkthrough Environment (2003) by Chim, J., Lau, R. W. H., Leong, H. V. and Si, A.

Through the research that has been carried out on SOI, it is found that in order to create a complete and realistic virtual walkthrough for a large and complex environment, a large amount of photographs are needed, these large amount of photographs will require a huge memory space, insufficient memory space will lead to a lag in the virtual walkthrough system; the virtual scene might not display on time during the walkthrough and this will directly affect the quality of the virtual walkthrough. Thus compression, prefetching and also caching mechanisms will be needed to create a smooth virtual walkthrough for a large and complex environment. Research and reading also have been done on MPEG compression, prefetching and caching mechanisms to understand the fundamental concepts of MPEG compression techniques and algorithm which are crucial in the future work of this project. In order to ensure that the current project developed is flexible enough to add in the compression, prefetching and caching functionality in the future enhancement, it is found that developing the virtual walkthrough in OpenGL and C++ will give the greatest flexibility and more control to the developer to modify any current features just to fit in the new enhancement. Literature review and research stage is carried out throughout the project development phase to continuously find the best option in creating the virtual walkthrough and also to keep track of the latest technology and idea used in creating the virtual walkthrough. Discussion and consultations carried out from time to time with Mr. M. Nordin Zakaria on issues regarding virtual walkthrough has eased the learning path and enhance understanding on virtual walkthrough. All these have been helpful in constructing a virtual walkthrough which is as perfect as possible.

3.1.5 Design and construct prototype

i. Virtual Walkthrough architecture design

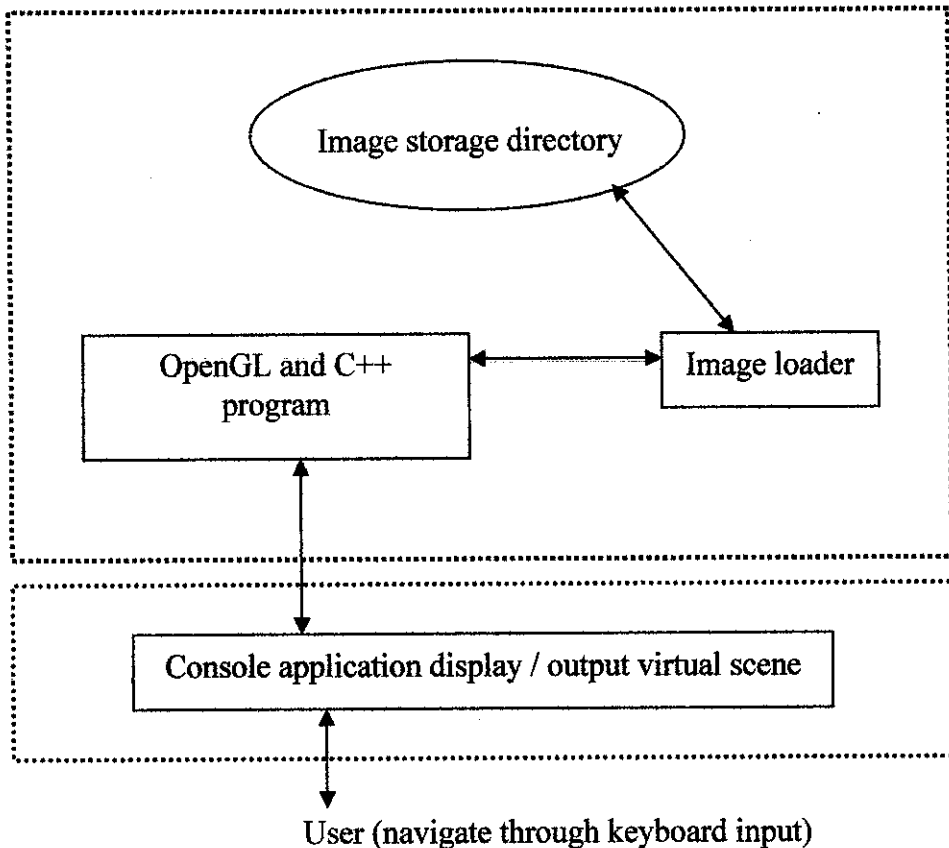


Figure 3.2 Virtual walkthrough architecture design

This project is intended to use low level computer graphics language such as OpenGL and C++ to create a two dimensional virtual walkthrough through the panoramic photographs taken. There are reasons why the low level programming language is more preferable compare to the high level computer graphics language, these are justify in Chapter 5. Figure 3.2 shows the architecture of the virtual walkthrough system, each of the module shows in Figure 3.2 is explained below.

Base on the virtual walkthrough architecture design in Figure 3.2, the viewer is able to walkthrough the virtual environment by using **keyboard** as the main navigation tool.

The **OpenGL C++ program** will then capture the viewer's keystroke and use the value of the keystroke to determine the current location of the viewer. By knowing the current location of the viewer, the photographs need to be retrieved to create the virtual scene can be determined. The virtual scene is created by using the texture mapping method, which will be explained in detail in Section 3.1.5.3.

The OpenGL C++ program will use an **image loader** to load the photographs needed to create the virtual scene. After knowing which photographs to load, the image loader will get the corresponding photographs from the image storage directory. The retrieved photograph is later loaded into the OpenGL C++ program and displays the virtual scene through the **console application window**.

The above steps will repeat every time the viewer moves or changes his or her position in the virtual environment by using the keyboard.

ii. Take Panoramic Photographs

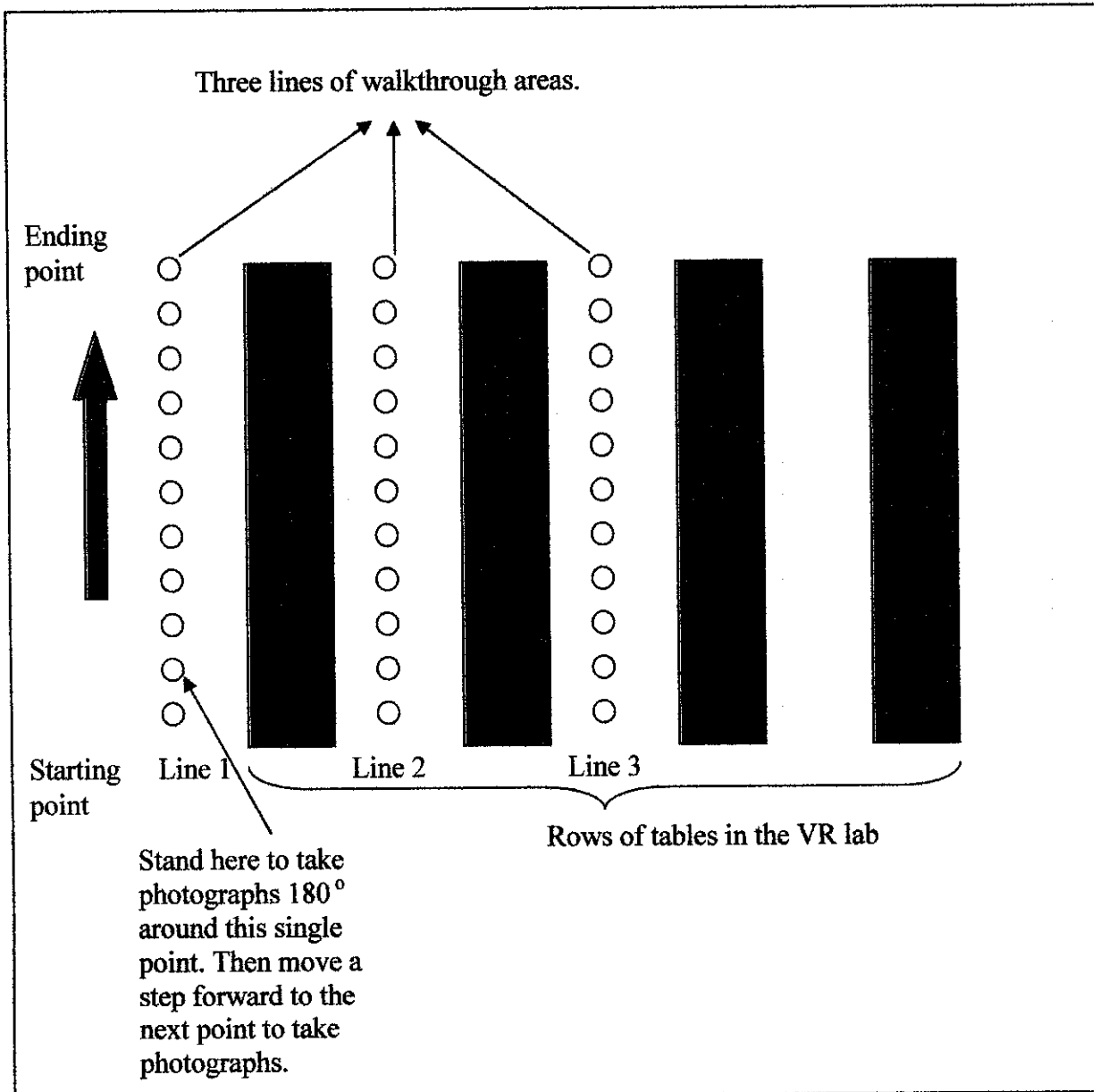


Figure 3.3 Steps to take panoramic photographs

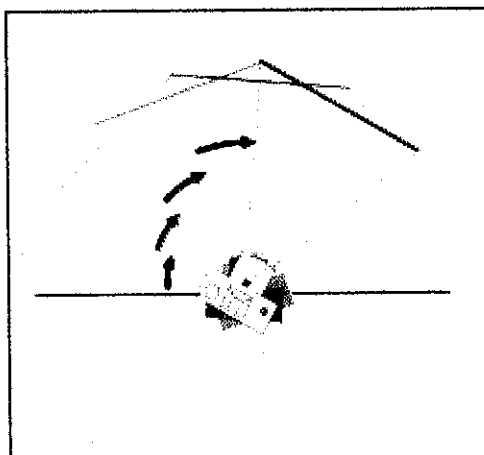


Figure 3.4 Capture images 180° around a single point of rotation

Since the virtual walkthrough is solely based on two dimensional images to create the two dimensional virtual scene, a dense of 1472 photographs with multiple direction are taken in the Virtual Reality (VR) lab. In order to implement the prototype, three lines of photographs are taken, as shown in Figure 3.3. At each line, photographs are acquired on an eye-height plane with every step taken without the used of a tripod. A tripod is not used in developing this prototype is because I want the virtual walkthrough to be as realistic as possible, as though a person is walking in the real world where there is tolerance on a little jerkiness in the walkthrough. At every step taken, many photographs are captured 180° around the single point of rotation with each of the photographs taken overlap each other by more than 50% horizontally; this is as shown in Figure 3.4. Figure 3.5 shows the result of the photographs taken 180° around the single point of rotation with each photographs overlap each other by more than 50% horizontally. No specific measurement is used to measure the exact distance need to be taken between each step and also no specific measurement is used to measure the exact angle need to be turn to take the subsequent overlapping photographs, all of the photographs are taken through approximation, this is because in this prototype I am more concern on the

outcome and the successfulness in creating a virtual walkthrough with multiple direction by using OpenGL and C++ approach rather than the accuracy or perfections of the virtual walkthrough.

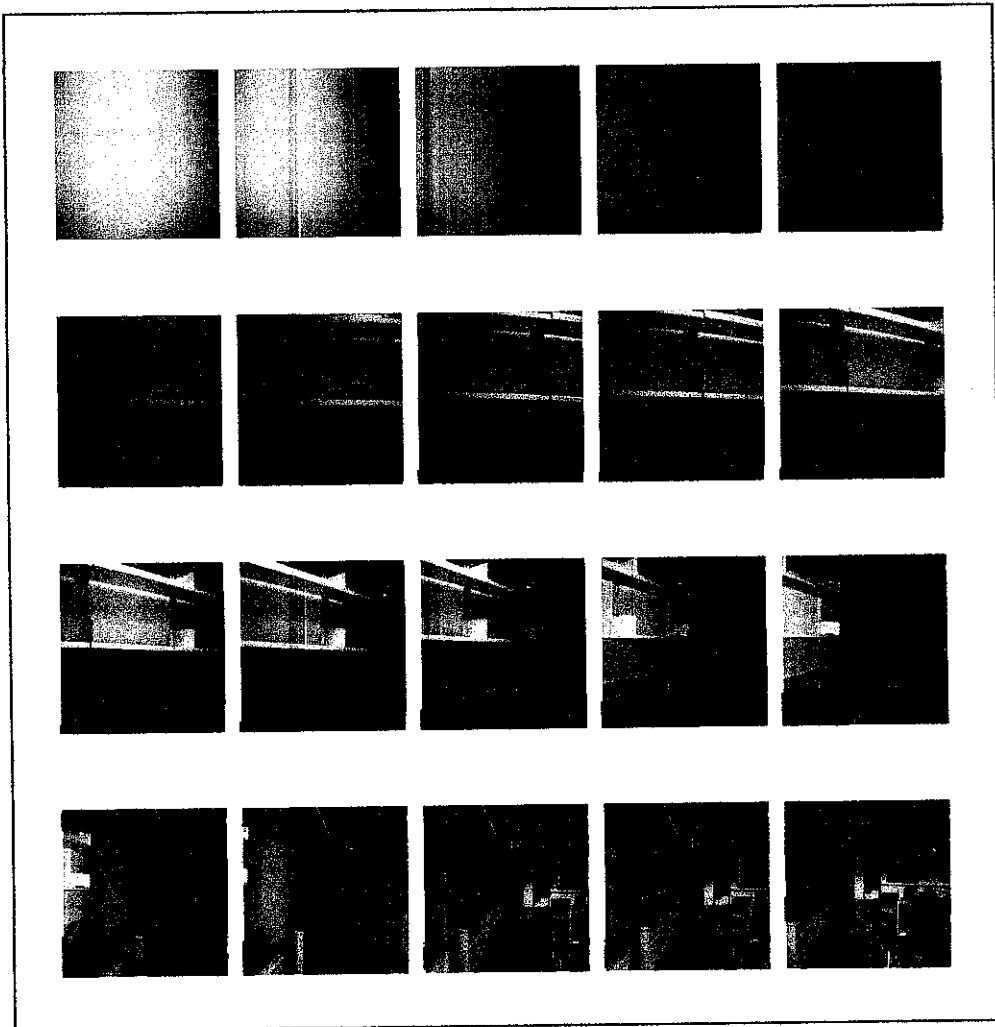


Figure 3.5 Photographs taken 180° around the single point of rotation with each photographs overlap each other by more then 50 % horizontally

The camera settings used to take the single direction sea of images are as below:

- White balance: Fluorescent FL2
- Flash: Without flash

Other settings remain unchanged (initial auto setting). No zooming is applied and the camera is hold horizontally in landscape orientation so that the captured photographs will have the maximize viewpoint. Please refer to Figure 3.6.

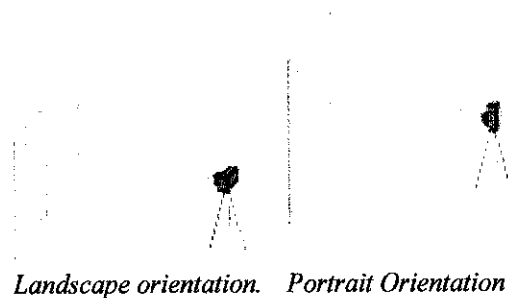


Figure 3.6 Camera orientation

iii. Convert the JPEG photographs captured into BITMAP format

The texture loader source file and header file used in the OpenGL program to load the texture will only take in BITMAP format file with the size of 256x256. The original JPEG format photographs are converted in batch into BITMAP format and resized by using Pic2Pic Plus freeware.

iv. Design the virtual walkthrough program

The idea and concept behind this virtual walkthrough system is to design a virtual walkthrough without using any game engine function or any built in QuickTime function but solely rely on the low level OpenGL and C++ functions. This is because in order to construct a complete walkthrough for a large complex environment in the future, a massive amount of photographs will

be needed. The massive amount of photographs will take up a lot of memory space and might slow down the system which leads to lack in presenting the virtual scene. The codes are written in OpenGL and C++ as they are more flexible and give more space for improvement in the future. The current OpenGL and C++ codes can be integrated with any prefetching and caching mechanism as well as MPEG compression and image interpolation codes in the future to compress the acquired data in a multiresolution hierarchy to enable the massive amount of captured data to be accessed efficiently to reconstruct the novel images during the virtual walkthrough. It is believed that with the prefetching and caching mechanism as well as MPEG compression and image interpolation functionality implemented in this system in the future, this could be a perfect two dimensional virtual walkthrough system for a large and complex environment.

The OpenGL and C++ programs develop for the virtual walkthrough is based on the following concepts:

- ***Keyboard as a user navigation device***

Six specific keys on the keyboard are programmed to enable the viewer to navigate in the virtual environment. These keys enable the viewer to move forward, backward, left, right and to have a 180° view at every step he or she takes. When the above mentioned keys are pressed, the OpenGL program will be able to capture the keystroke and use the value of the keystroke captured to determine the current location of the viewer. By knowing the current location of the viewer, the OpenGL and C++ program will be able to know which photographs need to be retrieved to create the virtual scene. Thus a command can be sent to the image loader to load the corresponding photographs from the image storage directory. The image loader command will be triggered every time the viewer pressed the navigation keys to move

to another location so that the corresponding virtual scene will be constructed.

▪ ***OpenGL texture mapping concept***

The two dimensional virtual scene is constructed based on the texture mapping concept. All 1472 photographs in the image storage directory are treated as a texture. A four sided polygon is created by using OpenGL, this polygon has the size same as the display screen and is placed exactly right on the display screen. So that the whole screen is covered by the texture photograph when a photograph is texture mapped onto the polygon, which will then gives the viewer a feeling that the virtual scene is changing according to the viewer's movement. Different photographs will be used to texture mapped onto the polygon, giving the virtual environment a change in scene depending on the viewer's location. No two subsequent movements will have the same virtual scene unless the viewer stay static in the same location and in this case which means the viewer does not press any key.

The first thing that must take place in the process of uploading the texture is a call to *glBindTexture*. *glBindTexture* tells OpenGL which texture "id" will be used. A texture "id" is just a number that use to access the textures.

This call will make texture that is associated with the "id" the active texture. Any call to OpenGL texture mapping will affect this texture.

Then enable the texture and bind the texture on the specify coordinate on the polygon.

The *glTexParameteri* sets the various parameters for the current OpenGL texture. Each of these lines are important and should not be left out.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

The *glTexEnvf* call sets environment variables for the current texture. It tells OpenGL how the texture will act when it is rendered into a scene.

glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL); is used to specify that the original texture color should be maintained and no lighting effect should be applied.

- ***Multidimensional arrays***

To be able to detect the viewer's location and retrieve the correct photographs to texture mapped the virtual environment; three dimensional array, `array3D[i][j][k]` are used to keep track of the viewer's location; *i* is to keep track of which line the viewer at, *j* is to keep track which position and *k* is to keep track of the angle since the viewer is able to view 180 degree at each position. The value of *i*, *j* and *k* will always change, depending on the location of the viewer, the value of *i*, *j* and *k* will either increase or decrease as the viewer navigates by pressing the key on the keyboard. `array3D[i][j][k]` with *i* = 1, *j* = 3, *k* = 10, means that the viewer is at line number 1, position number 3 with the angle of 10; thus image at line 1, position number 3 with the angle of 10 needs to be loaded, please refer to

Figure 3.7. In order to differentiate the massive amount of photographs and load the correct one, the name of the BITMAP photographs are name in almost the same way as the three dimensional array. As an example, the photographs taken at location line 1 position number 3 and angle 10 is named as 1_3_10.bmp, thus the correct photographs can be retrieved easily by knowing the correct location of the viewer.

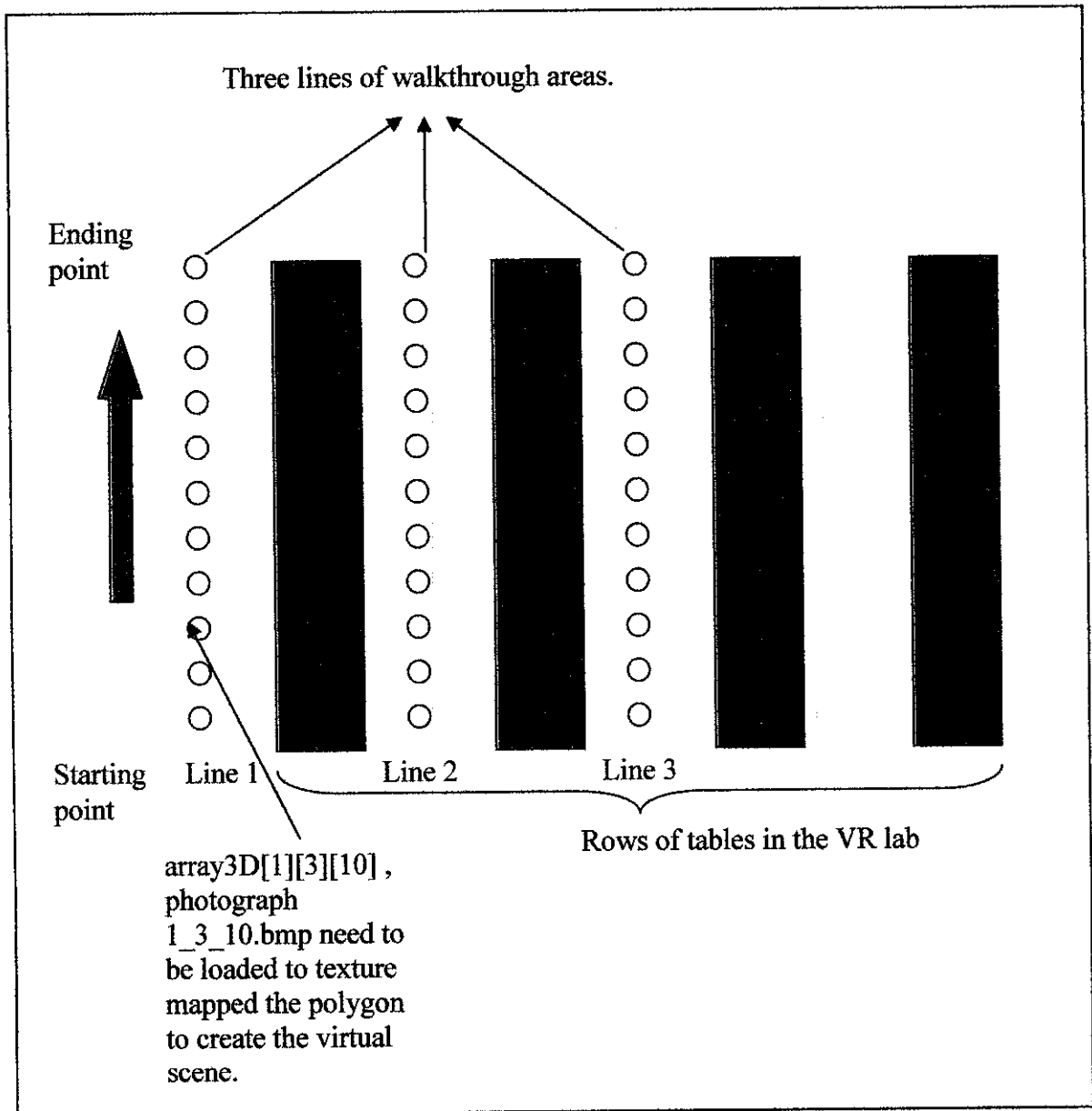


Figure 3.7 Usage of multidimensional array

3.1.6 Evaluation of the system

After the virtual walkthrough prototype has been constructed, evaluations are conducted on various aspects to measure the quality of the virtual walkthrough created. Evaluations are carried out on the memory storage consumed by the virtual walkthrough prototype, frame rate of the virtual scene display, smoothness of the virtual walkthrough as well as the quality of the virtual scene display. The findings of the evaluations are discussed in Chapter 4.

3.2 Tools Required

3.2.1 Digital Camera

The main source material in implementing the virtual walkthrough is photographs, thus it is crucial in selecting the type of digital camera which can produce the sets of photographs with the desired quality. This is very important to ensure that the virtual walkthrough constructed has good display quality. The selected digital camera is Nikon COOLPIX 5900, which has 3x zoom and has the capability of capturing digital photographs with 5.1 Megapixels resolution. Tripod is not used when capturing the photographs. The quality of the photographs captured is good enough to be used in the virtual walkthrough prototype.

3.2.2 Hardware

No high performance or super computer is needed to implement the virtual walkthrough prototype. The whole implementation work of the virtual walkthrough prototype is implemented on a personal computer installed with only Microsoft Windows XP Professionals Version 2002 Service Pack 1 operating system. The processor used is Intel Pentium IV CPU, 1.70 GHz with 224MB of Ram and a hard disk space of 38.1 GB.

3.2.3 Software

Photo Based 3D Walkthrough is created in Microsoft Visual Studio.Net 2003 Visual C++ project in the form of console application. The language used to create the walkthrough is OpenGL and C++ programming language. To be able to load the BITMAP image files to texture mapped the polygon of the virtual scene; another two pieces of C++ programs are integrated with the virtual walkthrough program. The first C++ program contains the BITMAP file functions needed to load the BITMAP format image file while the second C++ program contains the BITMAP file definition of the BITMAP files data structure needed by OpenGL.

To ensure that the OpenGL program can be compiled and run successfully, header files such as GL.h, GLU.h, GLAUX.h and GLUT.h are added into the include folder in Vc7 of Microsoft Visual Studio.Net 2003 file. Libraries such as glut32.dll and Glut32.dll are added into WINDOWS system32 folder.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Project prototype

A prototype of the photo based 3D walkthrough has been implemented using OpenGL and C++ in Microsoft.Net framework. The walkthrough area which is covered by this prototype is the Virtual Reality (VR) lab of Universiti Teknologi Petronas (UTP). As mentioned in Chapter 3, only three lines of photographs at the pathway between the computer tables in the VR lab are used to construct the virtual walkthrough. During the virtual walkthrough, the viewer is able to move forward, backward, move left or move right to go to the next pathway, and turn around on the spot to have a 180 degrees view of the surrounding. Below are some of the screen shots took during the virtual walkthrough, for more images please refer to Appendix I and Appendix II.

4.1.1 Move forward and move backward

In order for the viewer to move forward and backward freely in the virtual environment, the OpenGL and C++ program has been programmed in a way that when the viewer pressed the up arrow, it will move forward and when the viewer pressed the down arrow, it will move backward. Figure 4.1 till Figure 4.4 shows the screen shot took during the virtual walkthrough, where the virtual scene will change according to the viewer's position, for more images please refer to Appendix I; as the viewer pressed the up arrow, which means he or she is moving forward, thus the screen need to be textured with the photographs took after the current position. Where else if the viewer pressed the down arrow, which means he or she is moving backward, the photographs took before the current position will be loaded to texture mapped the screen.

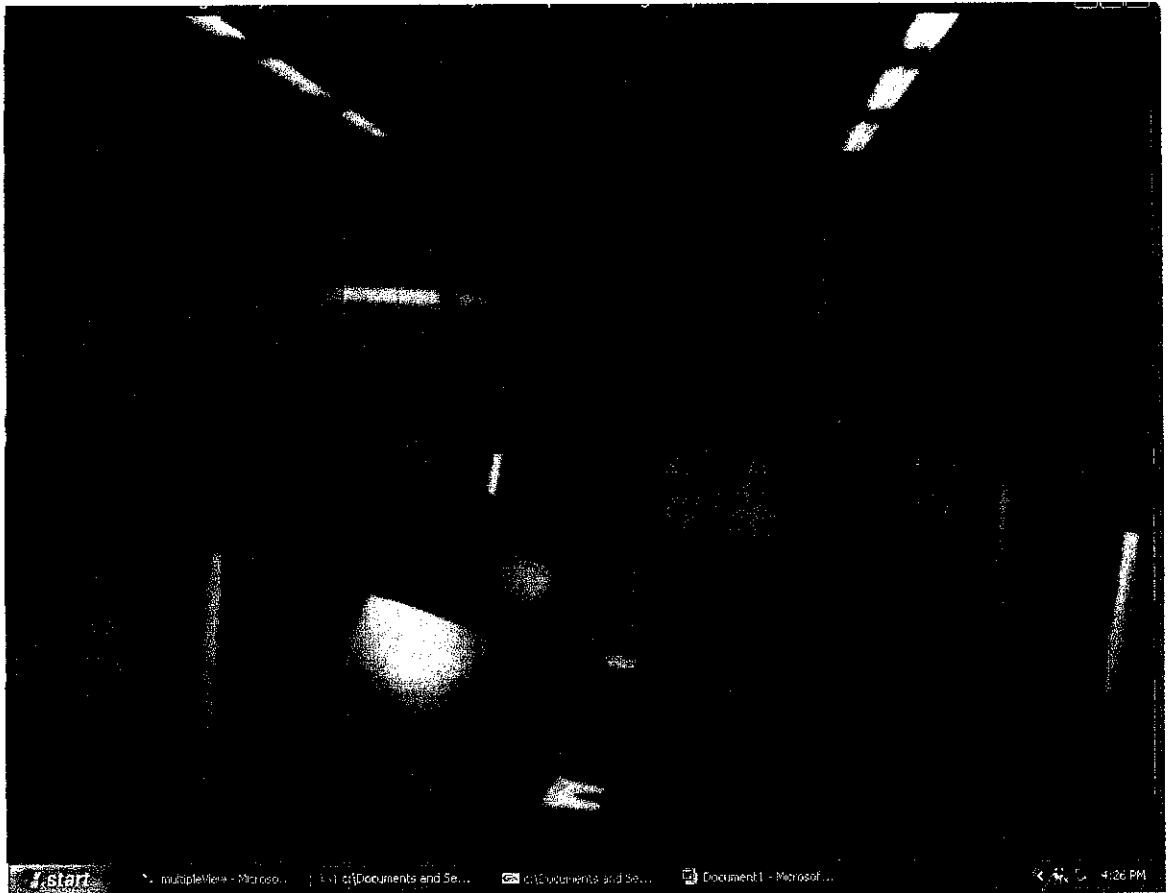


Figure 4.1 Moving forward and backward (1)

If the viewer currently is at the position in Figure 4.1, and wishes to move forward, he or she needs to press the up arrow. Observe the virtual scene in Figure 4.1, there is six computers on the computer tables located on the right hand side. As the viewer moves forward by pressing the up arrow, the virtual scene is Figure 4.2 will be displayed.

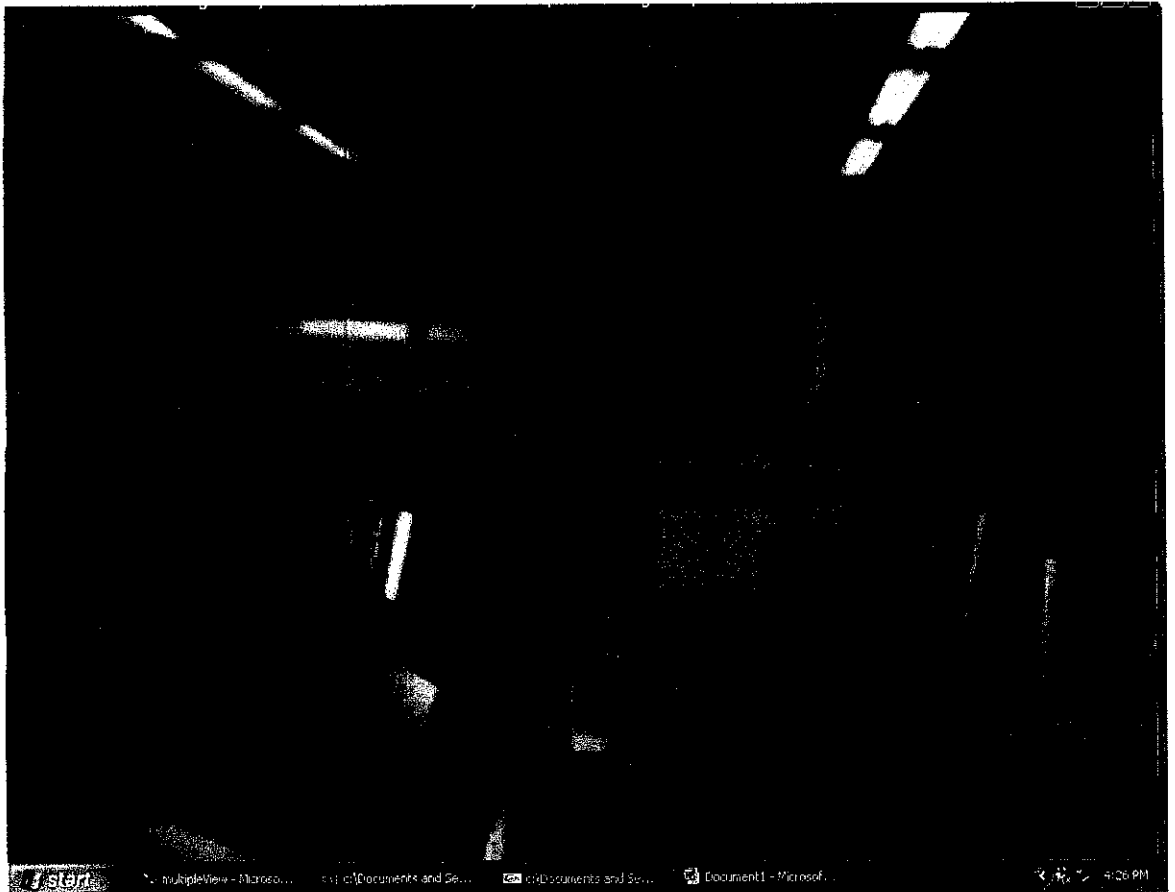


Figure 4.2 Moving forward and backward (2)

Observe the virtual scene in Figure 4.2. There are still six computers on the computer tables resided on the right hand side, but for the first computer which is the closest to the viewer, in this virtual scene, only the edge of the computer monitor can be seen; where else in the precious virtual scene in Figure 4.1 the whole computer monitor can be seen clearly. This indicates that the viewer has moved forward and is nearer to the first computer on the right.

If the viewer wishes to move forward continuously, then pressed the up arrow continuously, a series of photographs will be loaded, the virtual scene will change accordingly.

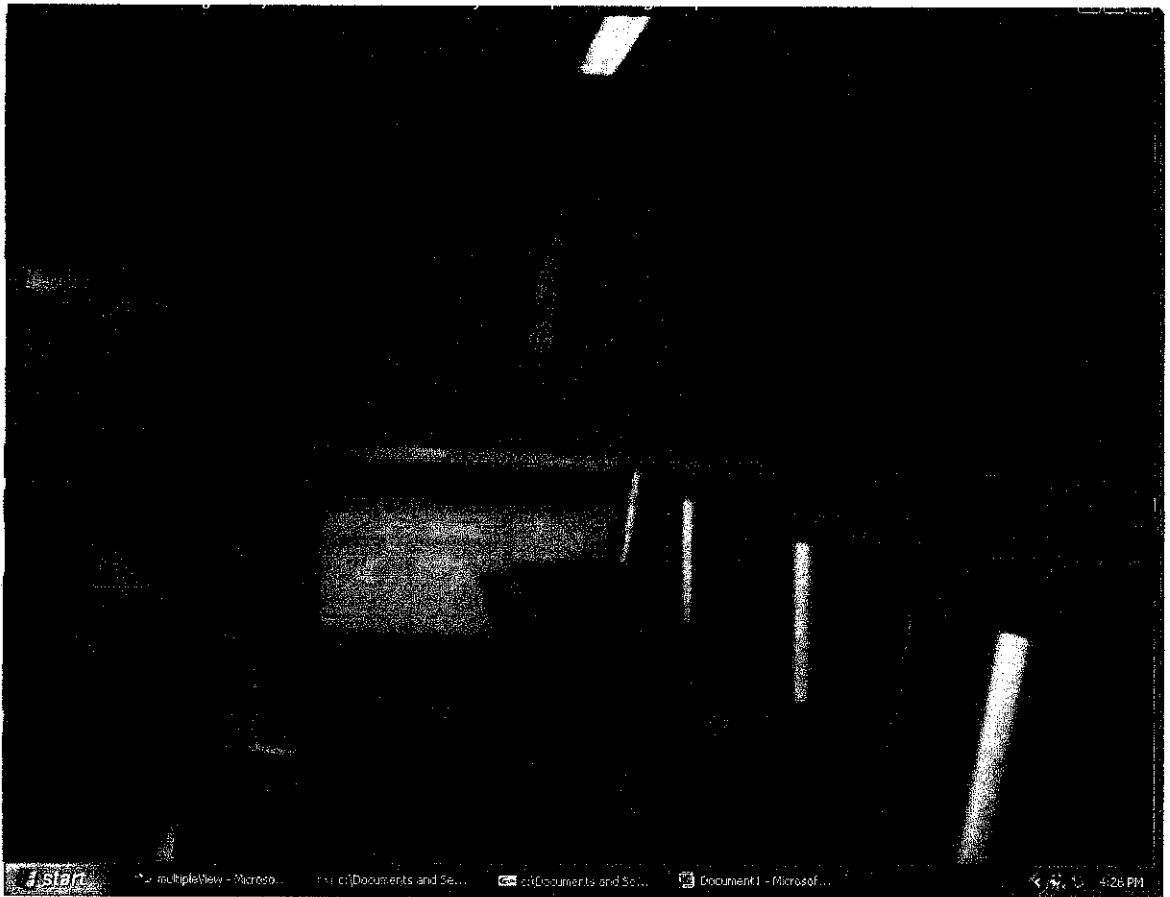


Figure 4.3 Moving forward and backward (3)

Figure 4.3 shows that the viewer has move more steps forward, where now only four computers can be seen on the computer tables resided on the right hand side. The viewer can move forward continuously by kept pressing the up arrow key until he or she reaches the end of the pathway.

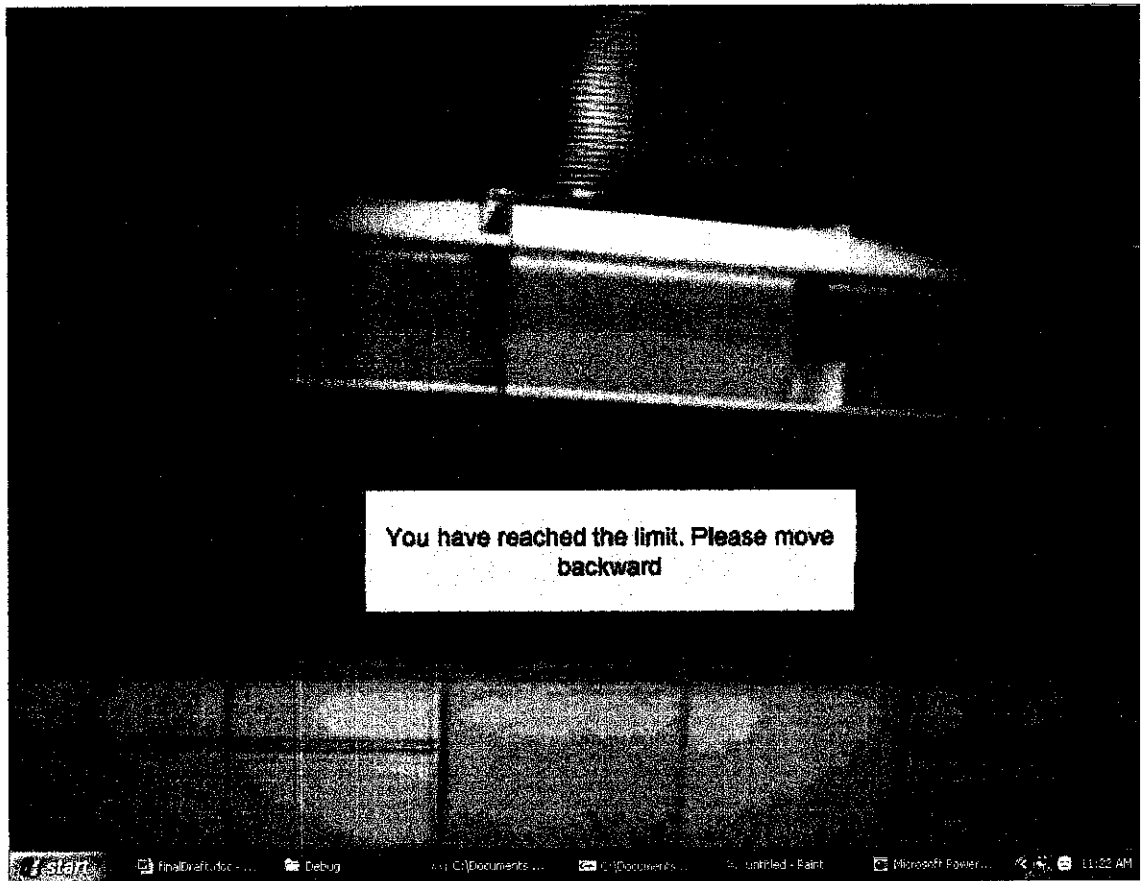


Figure 4.4 Alert message (1)

To guide the viewer during the virtual walkthrough, an alert message will be displayed to inform the viewer that he or she has reached the end of the pathway. Figure 4.4 shows the screen shot where the viewer has reach the end of the pathway.

In another case, if the viewer currently is at the position in Figure 4.3, and wishes to move backward, all he or she needs to do is just press the down arrow key, then the virtual scene at the previous location will be loaded, which gives the viewer an illusion that he or she is actually moving backward. If the viewer wishes to move backward continuously, then pressed the down arrow continuously, a series of

virtual scene from Figure 4.2 till Figure 4.1 will be appeared. The viewer is able to move backward continuously until it reaches the front most position of the pathway.

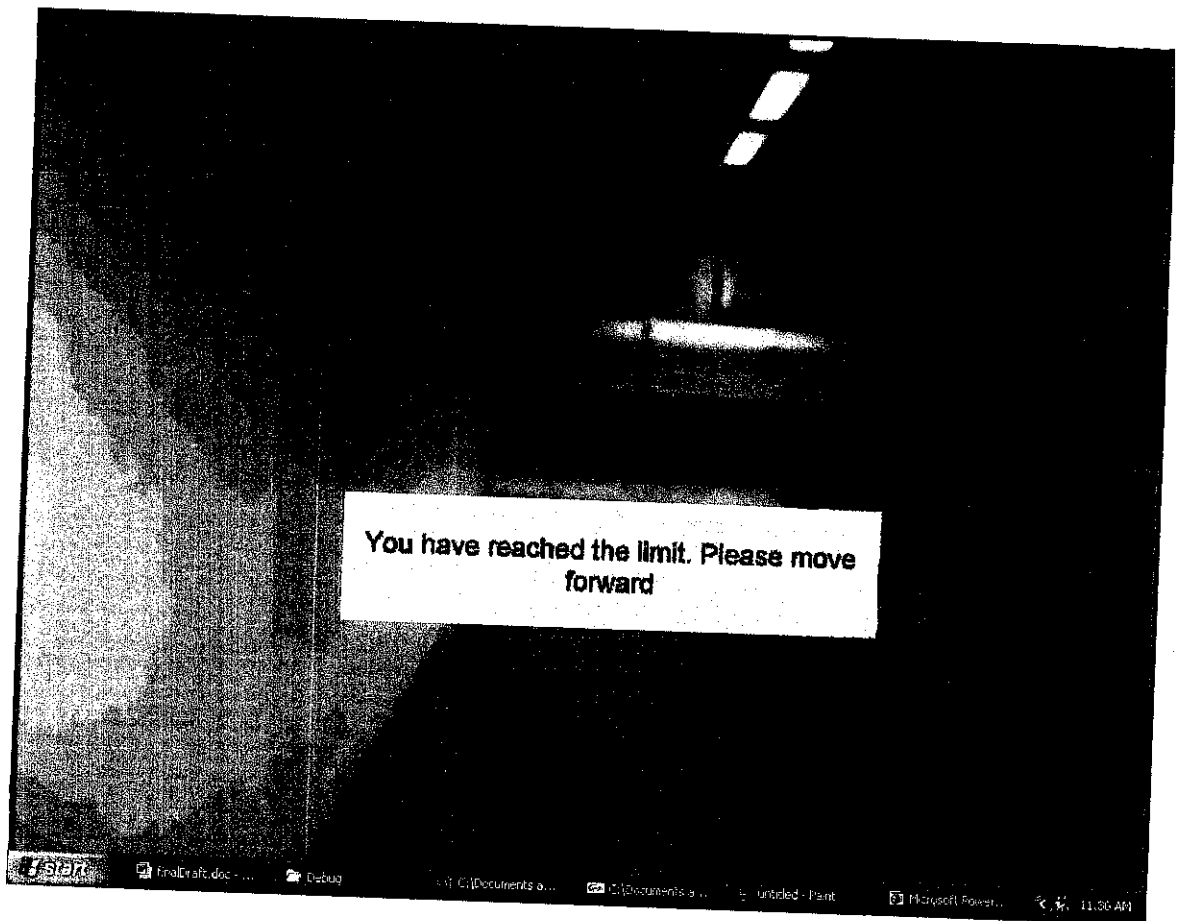


Figure 4.5 Alert message (2)

To guide the viewer during the virtual walkthrough, an alert message will be displayed to inform the viewer that he or she has reached the front most of the pathway. Figure 4.5 shows the screen shot where the viewer is at the front most position of the pathway.

4.1.2 Move left and right to go to the next pathway

In order for the viewer to be able to move to the left pathway or move to the right pathway between the computer tables in the VR lab easily, the OpenGL and C++ program has been written in such a way that when the viewer pressed F1, the viewer will move to the next pathway on the left, and when the viewer pressed F2, it will move the next pathway on the right.



Figure 4.6 Move left and right (1)

Figure 4.6 shows that the viewer is currently located at the position which is at the left most pathway, if the viewer pressed F2, he or she will move to the next pathway on the right, which is as shown in Figure 4.7.

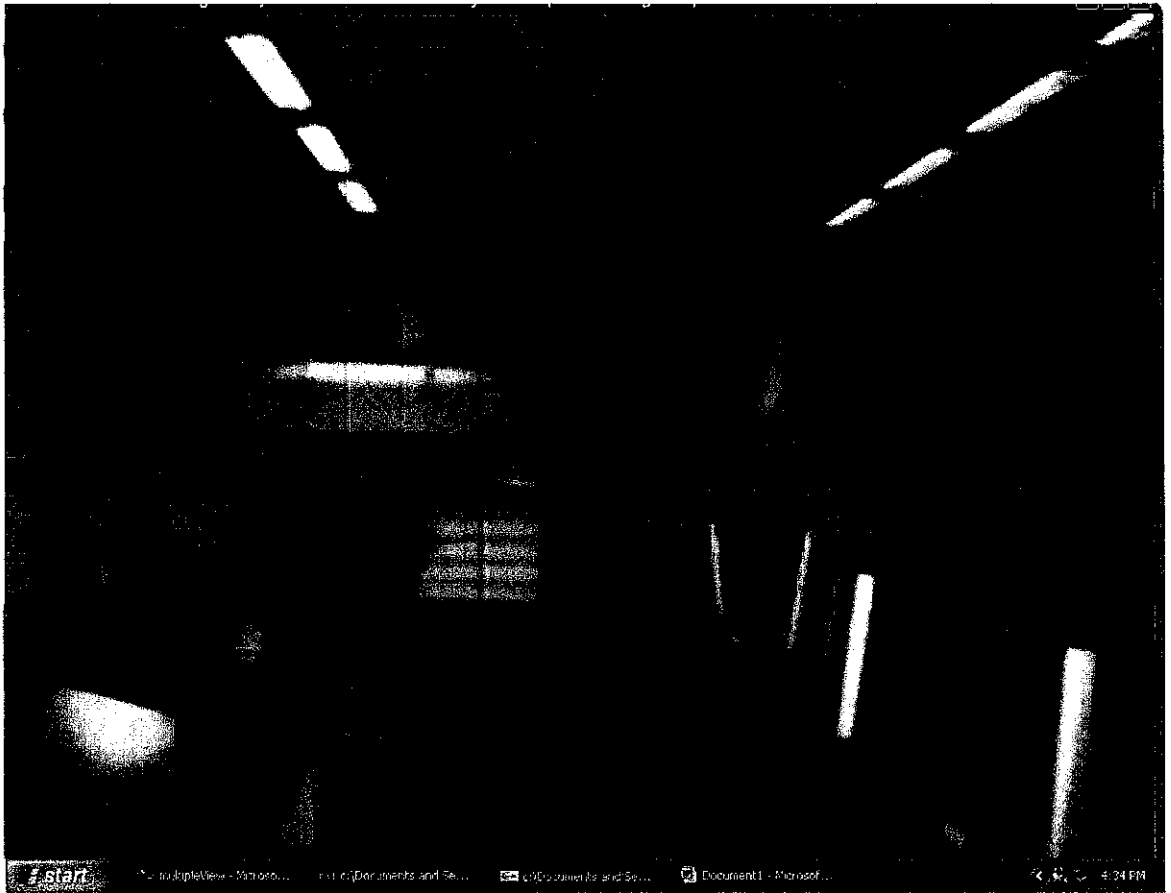


Figure 4.7 Move left and right (2)

Figure 4.7 shows that the viewer is now located at the second pathway. If the viewer wishes to move back again to the previous pathway (the first pathway, as shown in Figure 4.6), all he or she needs to do is just press F1.



Figure 4.8 Move left and right (3)

Since only three lines of walkthrough have been constructed, the third pathway is the right most pathways that a viewer can move to, as shown in Figure 4.8.

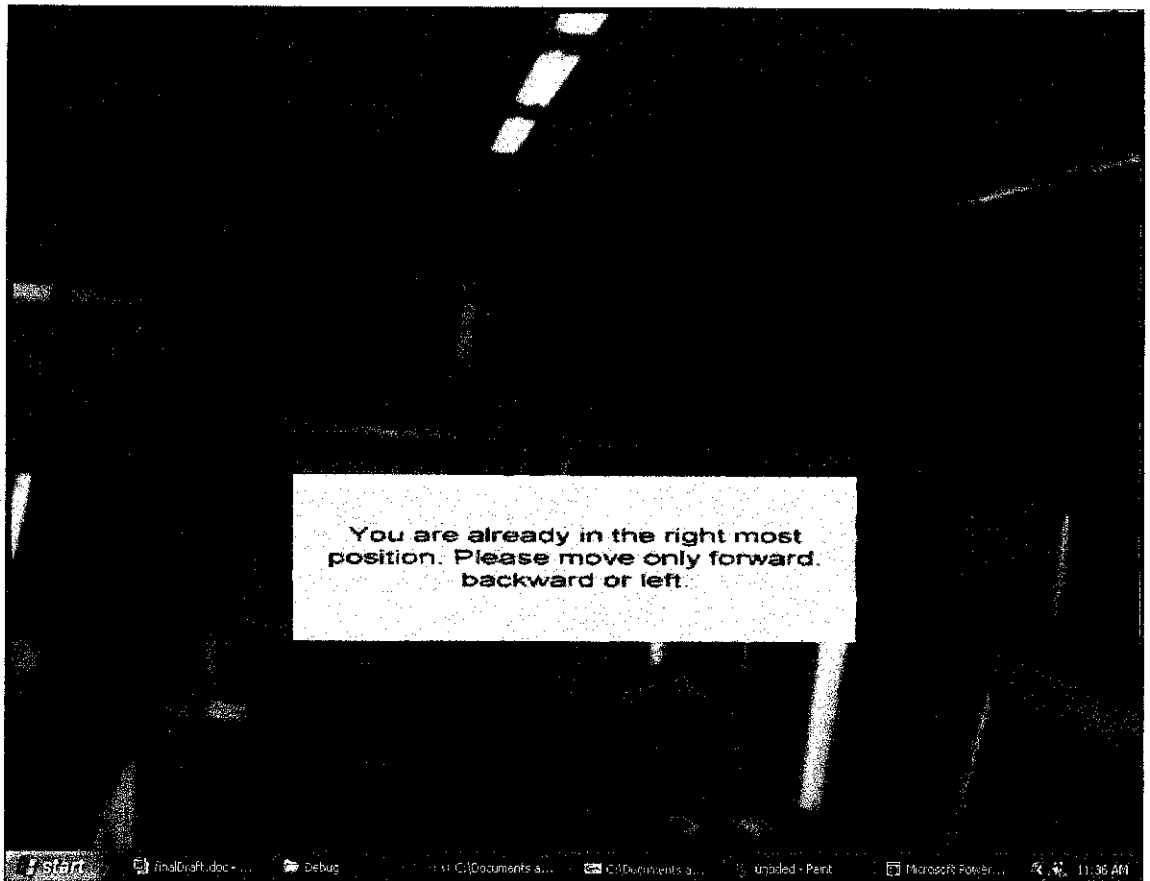


Figure 4.9 Alert message (3)

If the viewer press F2 again to move to the next pathway on the right (the fourth pathway), an alert message will be displayed to inform the viewer that he or she is already at the right most pathway. Figure 4.9 shows the screen shot where the viewer is at the third pathway (the right most pathways in this virtual walkthrough prototype) and try to move to the fourth pathway by pressing F2.

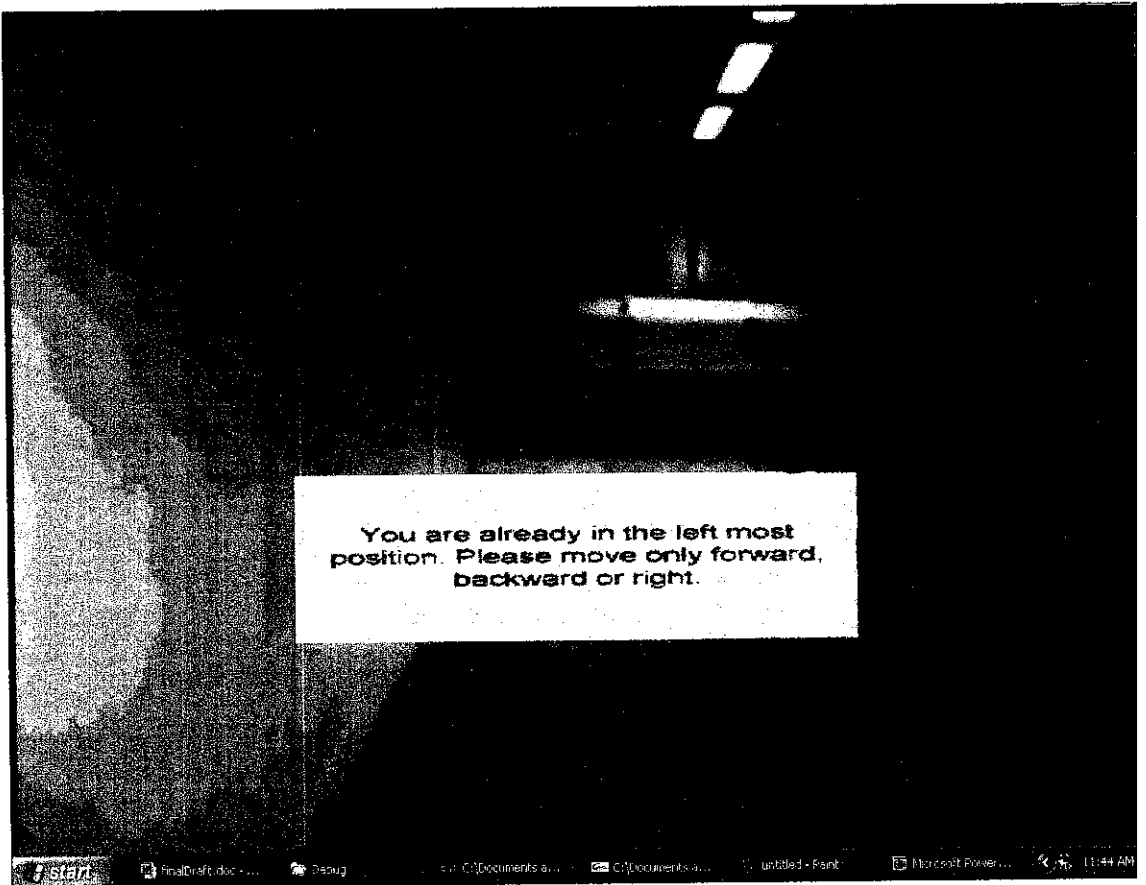


Figure 4.10 Alert message (4)

Same goes to when the viewer is already at the left most pathway, if the viewer press F1 again to move to the next pathway on the left, an alert message will be displayed to inform the viewer that he or she is already in the left most pathway. The viewer is not allowed to move to the left anymore because there is a wall on the left. Figure 4.10 shows the screen shot where the viewer is at the first pathway (the left most pathways in this virtual walkthrough prototype) and try to move to the left by pressing F1.

4.1.3 Turn around at a fixed spot to have a 180 degrees view at the surrounding

At every step taken, the viewer is able to turn around 180 degrees on the spot to view the surrounding. The viewer can pressed the right arrow key to turn to the right or press the left arrow key to turn to the left. Figure 4.11 till Figure 4.16 are part of the screen captured during the virtual walkthrough when the viewer is located at a fixed spot and he or she is viewing the surrounding from the fixed spot by pressing the right arrow key continuously until it reaches the end of the 180 degrees view at the right side. For more images please refer to Appendix II.

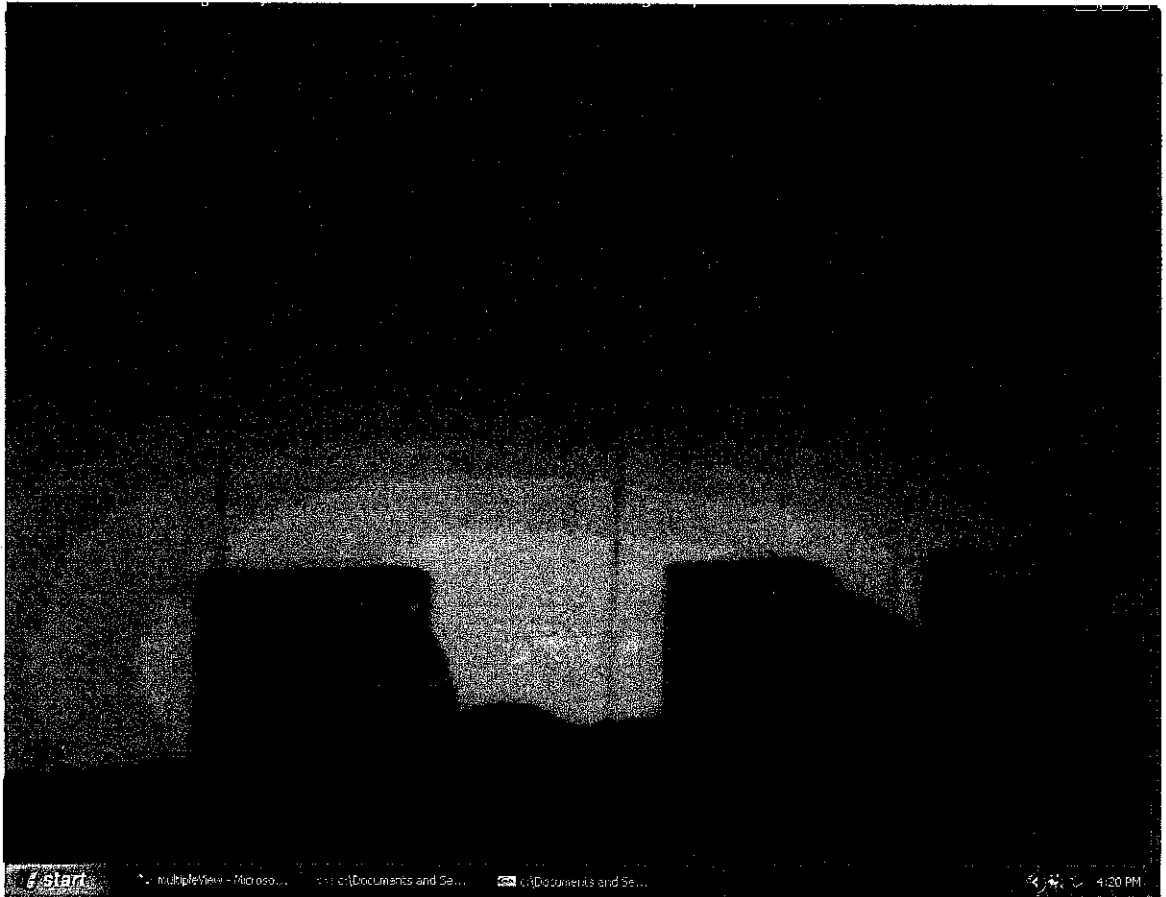


Figure 4.11 180 degrees view (1)

Figure 4.11 shows that the viewer is currently located at the third pathway and he or she is now looking at the left side from a fixed spot. In the virtual scene in Figure 4.11, there are three computers on the computer tables located at the left side. If the viewer wants to have a look at the surrounding at his or her right side, all the viewer has to do is just press the right arrow key. Once the viewer pressed the right arrow key, the virtual scene will change to the virtual scene as shown in Figure 4.12.

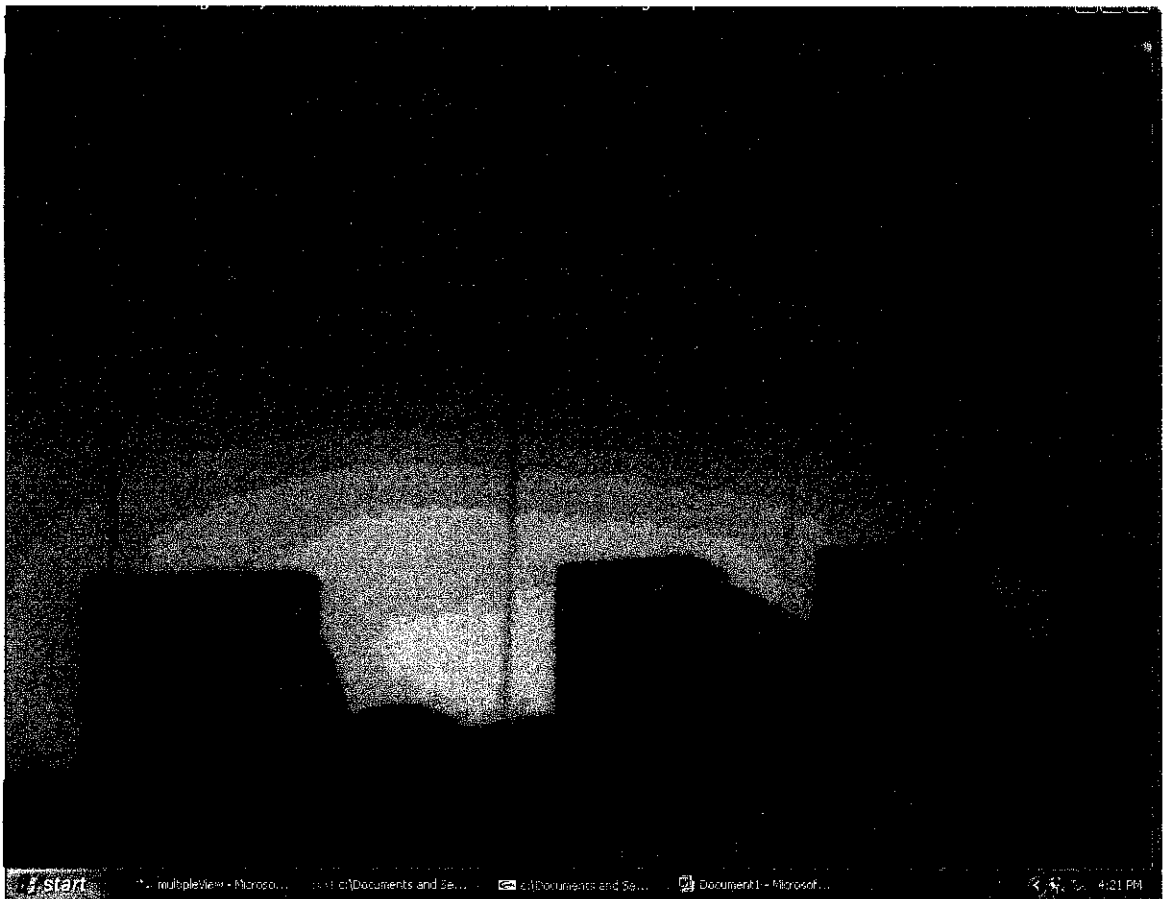


Figure 4.12 180 degrees view (2)

Observe Figure 4.12, it can be seen that now there are four computers appear in the virtual scene, The fourth computer cannot be seen just now from the angle in the previous virtual scene in Figure 4.11 which is more to the left side of the viewer.

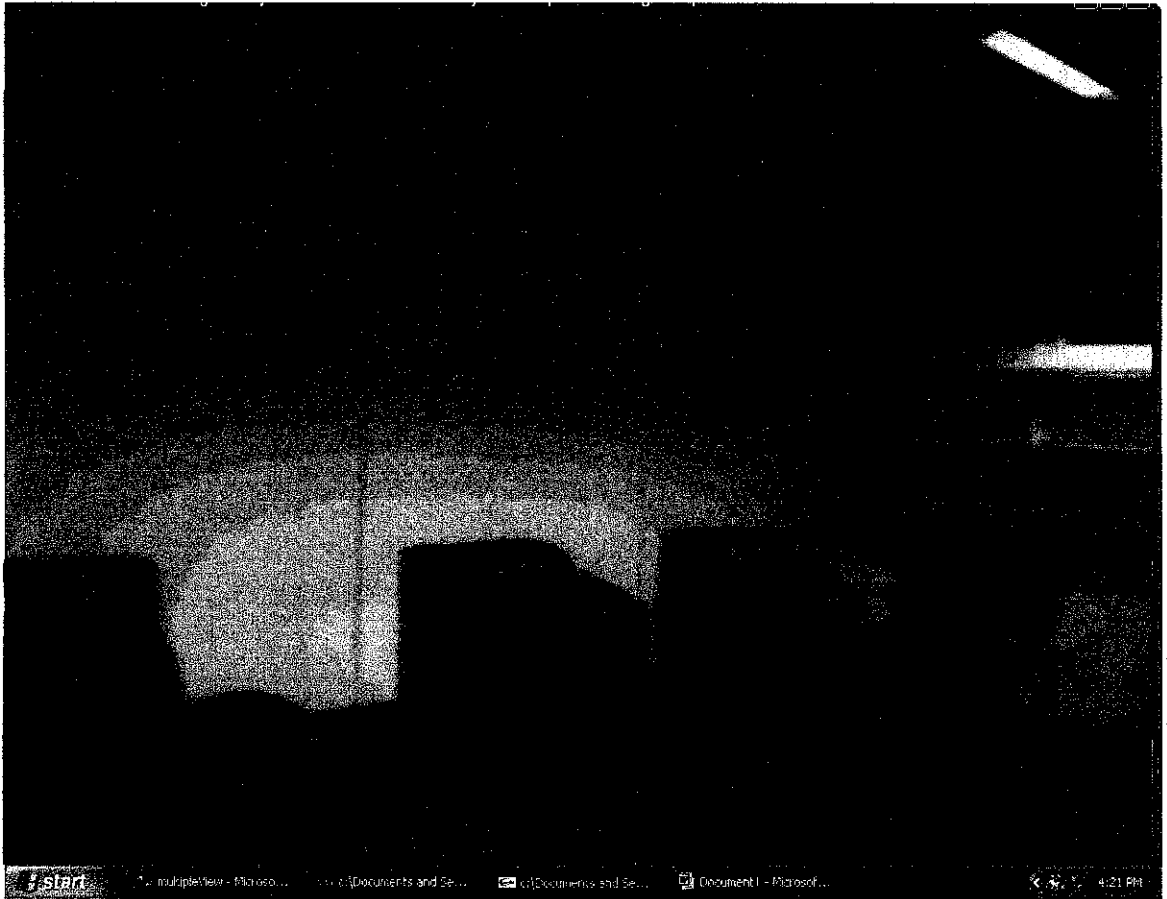


Figure 4.13 180 degrees view (3)

As the viewer turns more to the right side by pressing the right arrow key again, more view on the right side can be seen, as shown in Figure 4.13, more view after the fourth computer can be seen this time, a white cupboard can be seen at the back there clearly.



Figure 4.14 180 degrees view (4)

Figure 4.14 shows the virtual scene displayed when the viewer pressed the right arrow key again to look at the view on the right side. Now in this virtual scene, there are two computers can be seen on the computer tables in the second row. If the viewer wishes to turn back to the left again, all he or she needs to do is just press the left arrow key, the previous virtual scene, in this case virtual scene in Figure 4.13 will be displayed.

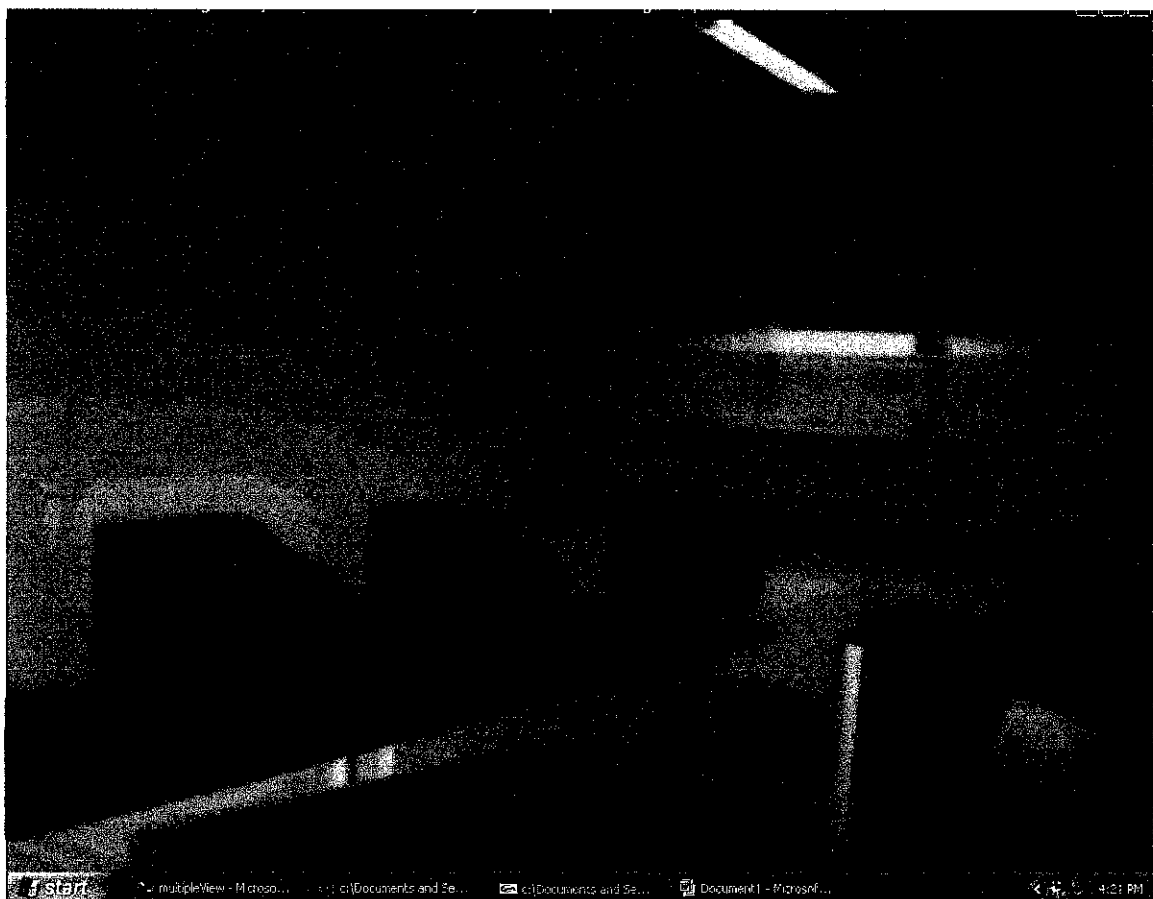


Figure 4.15 180 degrees view (5)

If the viewer kept pressing the right arrow key to look at the surrounding on the right hand side, the next virtual scene, in this case photographs in Figure 4.15 will be loaded to texture the virtual scene. As shown in Figure 4.15, more computers on the computer tables on the second row can be seen now.

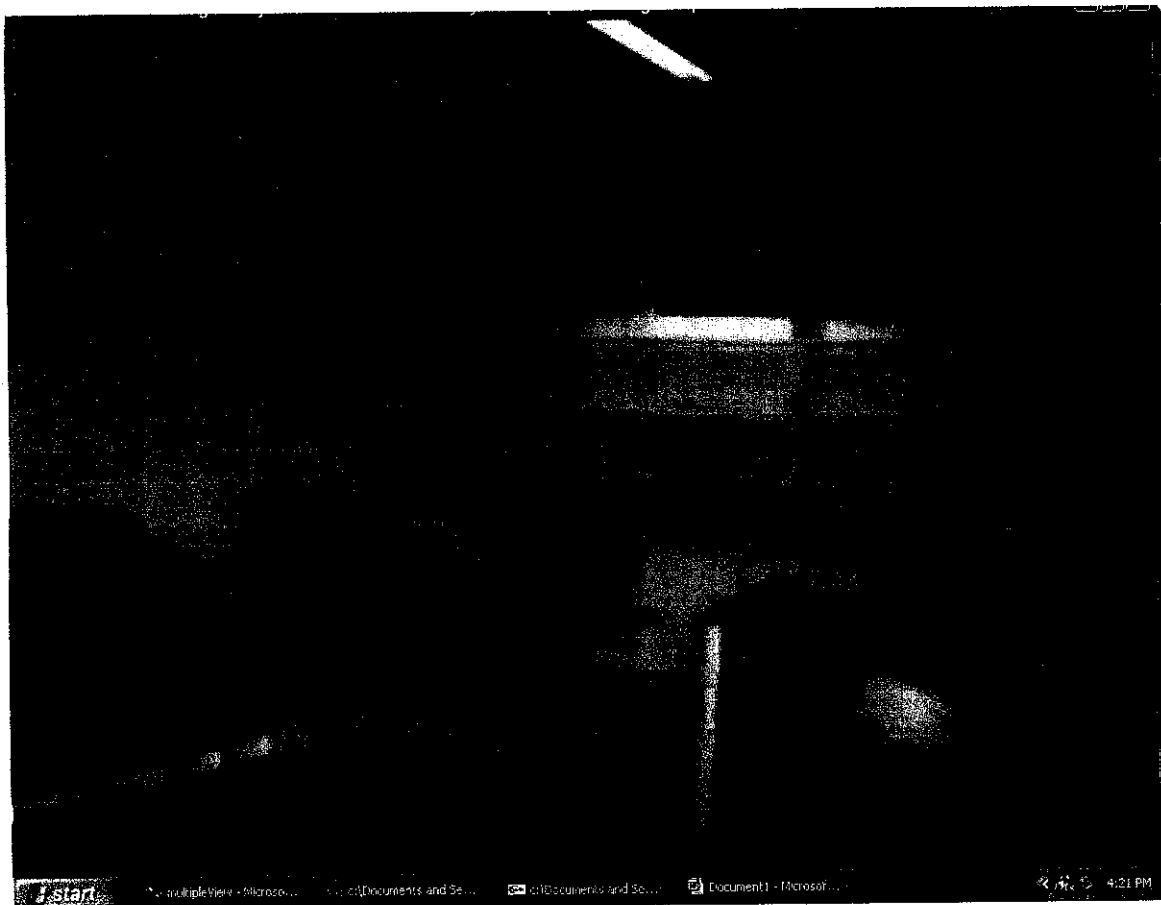


Figure 4.16 180 degrees view (6)

If the viewer still continue pressing the right arrow key, Figure 4.16 will be displayed. In Figure 4.16, more things on the right side can be seen now. There is a pole which is light brown in color towards the end of the second pathway can be seen in the virtual scene in Figure 4.16. The above process (pressing the right arrow key continuously) is as though the viewer is turning his or her head 180 degrees from the left looking to the right. To view to the left, all the viewers need to do is just press the left arrow key continuously, then this will be as though the viewer is turning his or her head 180 degrees from the right looking to the left. For more images, please refer to Appendix II.

4.2 Memory space consumption

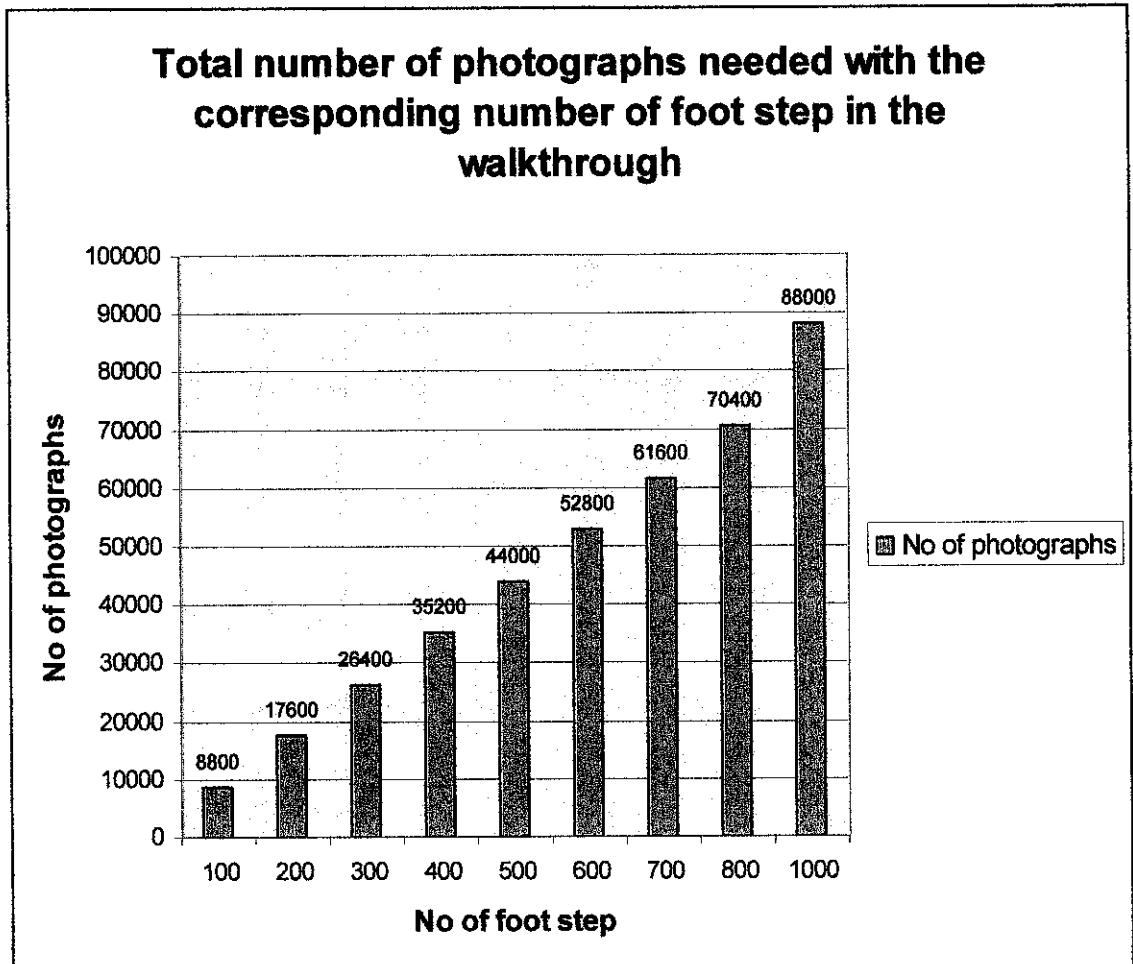


Figure 4.17 Total number of photographs needed with the corresponding number of footsteps

The walkthrough for the prototype constructed only cover three pathways between the computer tables in the VR lab, a total of 39 steps can be taken during the walkthrough; each with 180 degrees view. 1072 photographs are needed to construct the virtual walkthrough just for the three pathways only. Moreover, each step has only 180 degrees view but not 360 degrees view. The graph in Figure 4.17 shows the total number of

photographs needed to construct a virtual walkthrough with 360 degrees view for each step taken. As shown in the graph, to construct a walkthrough with 100 footsteps and each has 360 degrees view, 8800 photographs are needed.

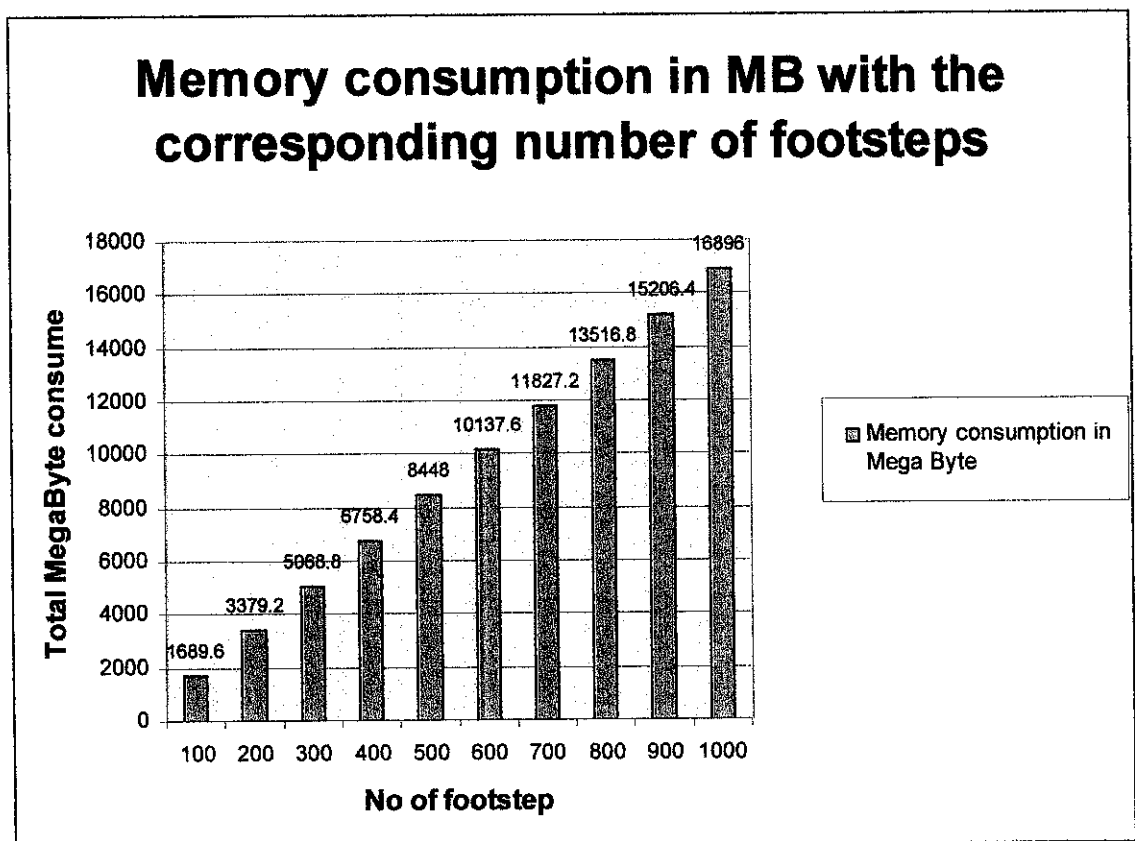


Figure 4.18 Memory consumption in MB with the corresponding number of footsteps

These 8800 photographs will consume a total number of 1689.6 MB (Megabyte), as shown in Figure 4.18. Both Figure 4.17 and 4.18 shows that the total number of photographs and total number of memory space consume increased directly with the total number of footsteps can be taken in the walkthrough. It is obvious that, in order to construct a virtual walkthrough for a large geographical area, the memory space

consumption will be very high. This problem will be more severe if a virtual walkthrough with 1000 footsteps were to be constructed; Figure 4.17 and Figure 4.18 show that a total of 88000 photographs which are 16896 MB (Megabyte) in size are needed. Due to the serious memory consumption problem, it is not practical to create a virtual walkthrough for a large geographical area. To solve this problem, a client and server architecture is recommended, where the server will store all the photographs and the client will retrieve the photographs from the server to texture the virtual scene during the walkthrough.

4.3 Prefetching Mechanism

After the completion of this project, a potential area of research and improvement has been found which leads to the continuation of this project. As discuss in Section 4.2, due to the large memory space consumption, client server architecture is needed to improve the virtual walkthrough system in order to create a virtual walkthrough for a large environment. The introduction of client server architecture in Photo Based 3D Walkthrough will enable the construction of a virtual walkthrough for a large geographical area, but on the other hand it creates latency in displaying the virtual scene. The client will continuously request for photographs from the server to create the virtual scene during the walkthrough. However, the requested photograph in the server need to be transferred across the network to the client before it can be used to display the virtual scene, transmitting images need quite a certain amount of time depending on the network load, thus creates latency in presenting the virtual scene. To encounter the latency problem, prefetching algorithm is needed to preload the photograph from the server to the client to eliminate the waiting time for the photograph to arrive at the client when it is needed to create the virtual scene.

In this project, there is no client and server architecture implemented. However, to simulate how prefetch can enhance virtual walkthrough in a client and server architecture, the windows Sleep() command is used to delay the system in displaying the virtual scene. Mean Prediction Scheme is used to estimate the next movement vector in order to predict the next location of the viewer. By knowing the next location of the viewer, the photograph at the next location can be prefetch while the viewer is still at the current position; therefore the waiting time is eliminated since the photograph is already preloaded to the client before hand.

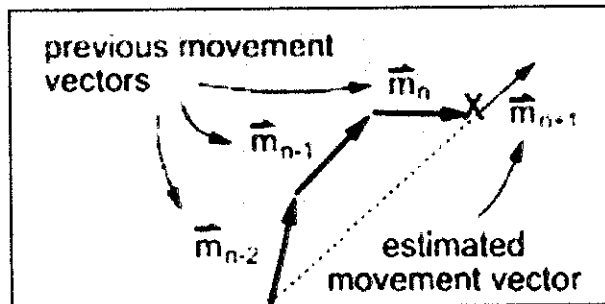


Figure 4.19 Mean Prediction Scheme (1)

The estimated movement vector of the viewer is calculated from the average of the previous movement vector. Figure 4.19 shows that the viewer is currently at the red X position, to estimate the next movement vector, \hat{m}_{n+1} ; the average of the previous movement vector is calculated, which is $\frac{\hat{m}_n + \hat{m}_{n-1} + \hat{m}_{n-2}}{n}$. From the estimated movement vector, the next location of the viewer can be predicted, which is indicated by the blue X in Figure 4.20.

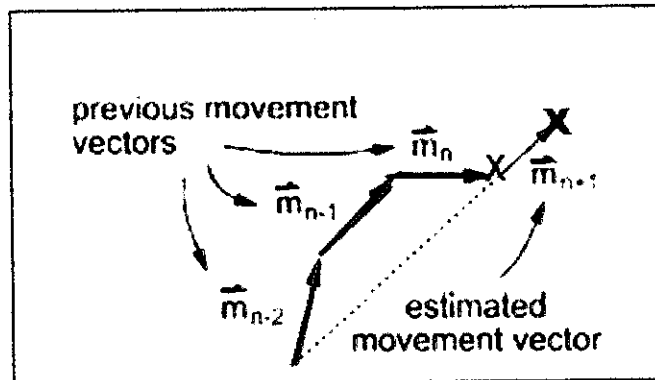


Figure 4.20 Mean Prediction Scheme (2)

Thus, while the viewer is still at red X location, the photograph at blue X location is already prefetch from the server to the client. As the viewer moves to the blue X location, the system will check whether the photograph at blue X location is already at the client. If it is already at the client, the virtual scene can be display immediately or else the system will sleep for some while to simulate the process of downloading the photographs at blue X location from the server.

With the implementation of the Mean Prediction Scheme, it does eliminate delay and improve system performance, but Mean Prediction Scheme does not adapt quickly to the changes in viewer's moving patterns. Mean Prediction Scheme is chosen to implement the prefetching mechanism is just for experimental purpose to experiment whether the client server system perform better with the prefetching mechanism, and also to test for the effectiveness of this scheme. Through observation, it is found that once the viewer changes the moving direction, the estimated movement vector will be inaccurate.

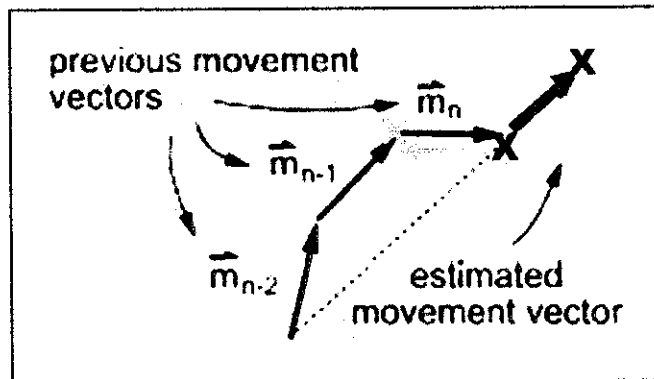


Figure 4.21 Mean Prediction Scheme (3)

Figure 4.21 shows that a viewer is currently located at blue X position. The red X position is the estimated next location, if the viewer changes his or her moving pattern and move to the green X position, then the estimated next location will be inaccurate. The photograph at red X position is prefetch previously, and when the viewer is at green X location, there is no photograph at green X location at the client. The client need to request the photograph at green X position from the server and therefore incurs latency.

4.4 Prefetching and Caching Mechanism

The Mean Prediction Scheme implemented for prefetching is not adequate to cater for sudden change in viewer's moving pattern. A Caching mechanism has been implemented to complement and enhance the Mean Prediction Scheme. A maximum total of eight textures are allowed to be binded at a time in OpenGL. This feature of OpenGL is made used to cache up till eight photographs of those positions nearest to the viewer's current viewing position.

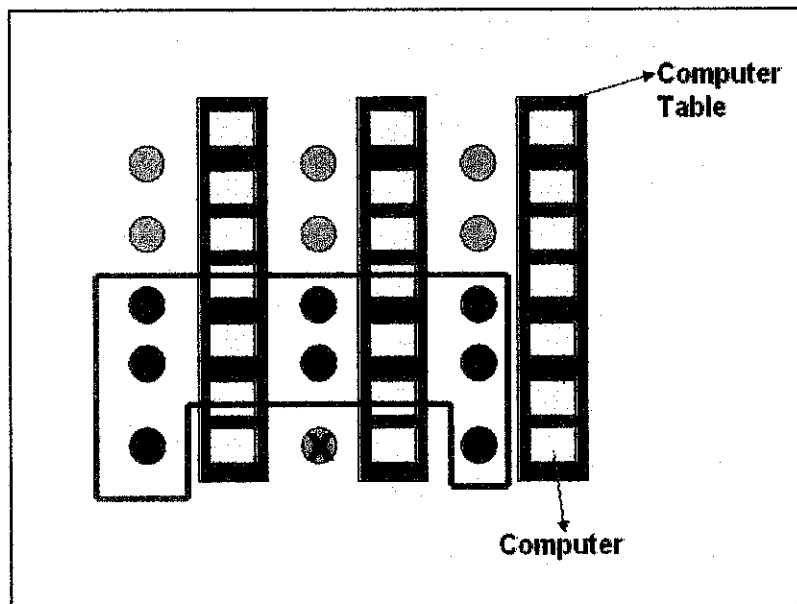


Figure 4.22 Caching (1)

Figure 4.22 shows that the viewer is currently located at the red X position. Therefore, the photographs at the nearest eight neighbor position, which is indicated by the red dot are cached into a cache table with each of them bind with a unique texture identifier. By doing so, the photographs at the closest eight position are already requested and transferred to the client when the viewer is at the red X location. As a result future

request and transfer time can be saved and latency is eliminated because the photographs are already available at the client to create the virtual scene.

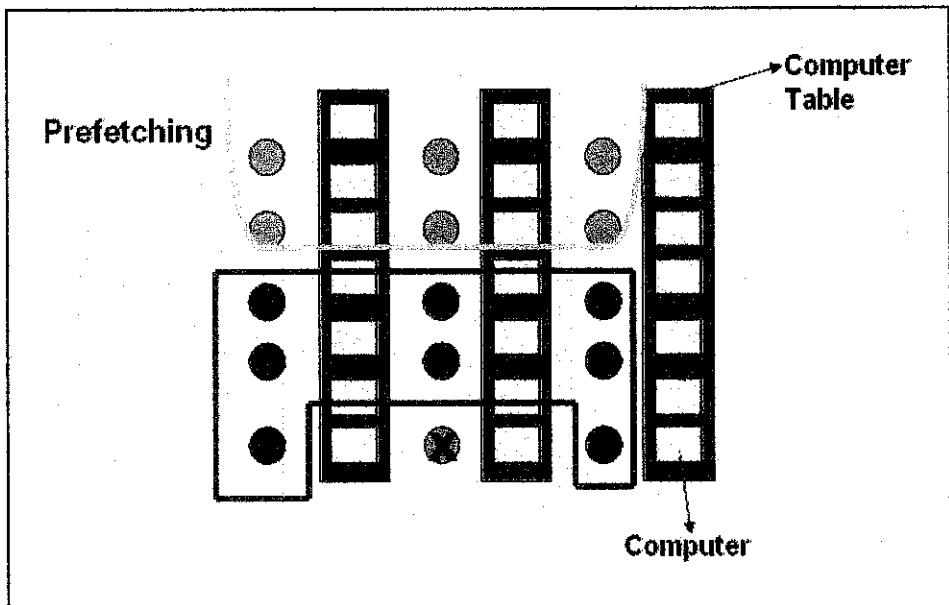


Figure 4.23 Caching (2)

When the viewer moves out of the cache boundary (indicated by the red line in Figure 4.23) the system will use the Mean Prediction Scheme discussed in Section 4.3. The green line in figure 4.23 shows the starting of the use of Mean Prediction Scheme to prefetch the photographs.

The caching mechanism implemented does not have update mechanism to update the cache table frequently. Thus, it will always bind the same eight photographs. The purpose of implementing the caching mechanism is to test whether it improves the walkthrough performance to decide on the area of further research to better enhance the system.

4.5 Comparison between JPEG and BITMAP format

The original format of the photographs captured is in JPEG format. However due to the texture loader source file and header file used in this OpenGL program to load the texture is in bitmap format, the JPEG format photographs must be converted into BITMAP format and resized to the width and height of 256 pixels. Efforts have been put in to construct a JPEG loader due to less memory storage consumed by JPEG file format which will be explained in the sections below, but it is not successful. Therefore, BITMAP loader is used in this prototype. The difference in quality as well as image file size between the JPEG format photographs and the BITMAP format photographs are discussed in Section 4.5.1 and 4.5.2.

4.5.1 Image quality

There is not much difference in the image quality perceived by human eyes between JPEG and BITMAP format. The difference in image quality is almost unnoticeable by human eyes; it is hard to differentiate whether the image is in JPEG or BITMAP format if the file format is not known. Below shows two photographs captured during the development phase of this project; one in JPEG format one has been converted into BITMAP format. The JPEG format photograph has the image quality of 8 while the BITMAP format has the depth of 24 bits. Both photographs have the same number of pixels value, 2592 x 1944. Please refer to Table 4.1.

Table 4.1 Image format and image quality

Image format	Image quality
JPEG	8
BITMAP	24 bits depth

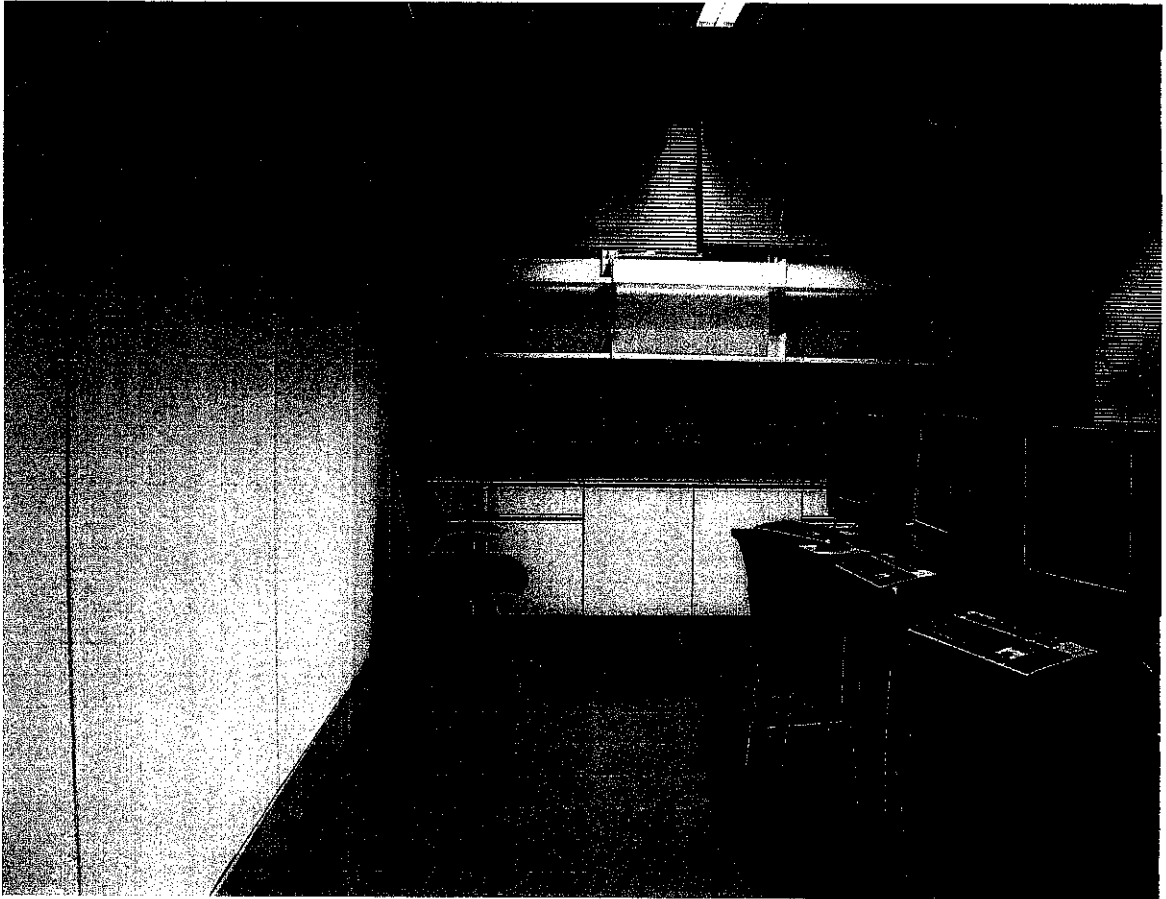


Figure 4.24 JPEG format photograph

Figure 4.24 above shows a photograph of JPEG format while Figure 4.25 below shows a photograph of BITMAP format. Differences in image quality perceived by human eye can be hardly seen in both images. Thus it makes no difference in image quality perceived by human eyes even if the JPEG image is converted into BITMAP format.

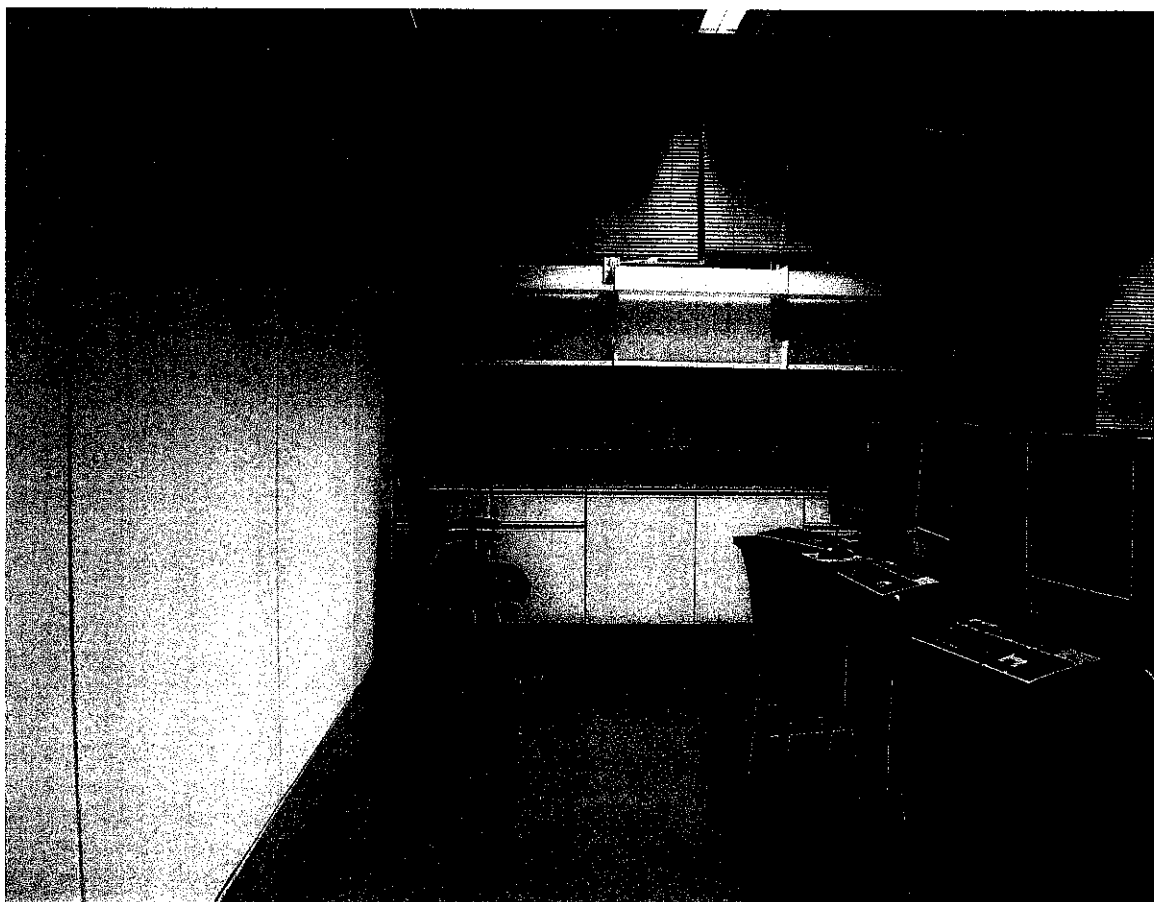


Figure 4.25 BITMAP format photograph

4.5.2 Comparison between JPEG and BITMAP image file size

There is a big difference between the image file size of a JPEG format photograph and a BITMAP format photograph. The image file size of a BITMAP format photograph tends to be a lot bigger than a JPEG format photograph. As a comparison, please refer to Table 4.2 which shows the difference in image file size for both BITMAP and JPEG format photograph which have the similar pixels value of 256 for both width and height.



Table 4.2 Image format and image file size

Image format	Image file size
JPEG	42.3KB
BITMAP	192KB

Table 4.2 shows that JPEG photograph has the image file size of 42.3KB while BITMAP photograph has the image file size of 192KB. There is a drastic difference in image file size between JPEG and BITMAP. An image which is in BITMAP format need five times more storage space compare to JPEG format.

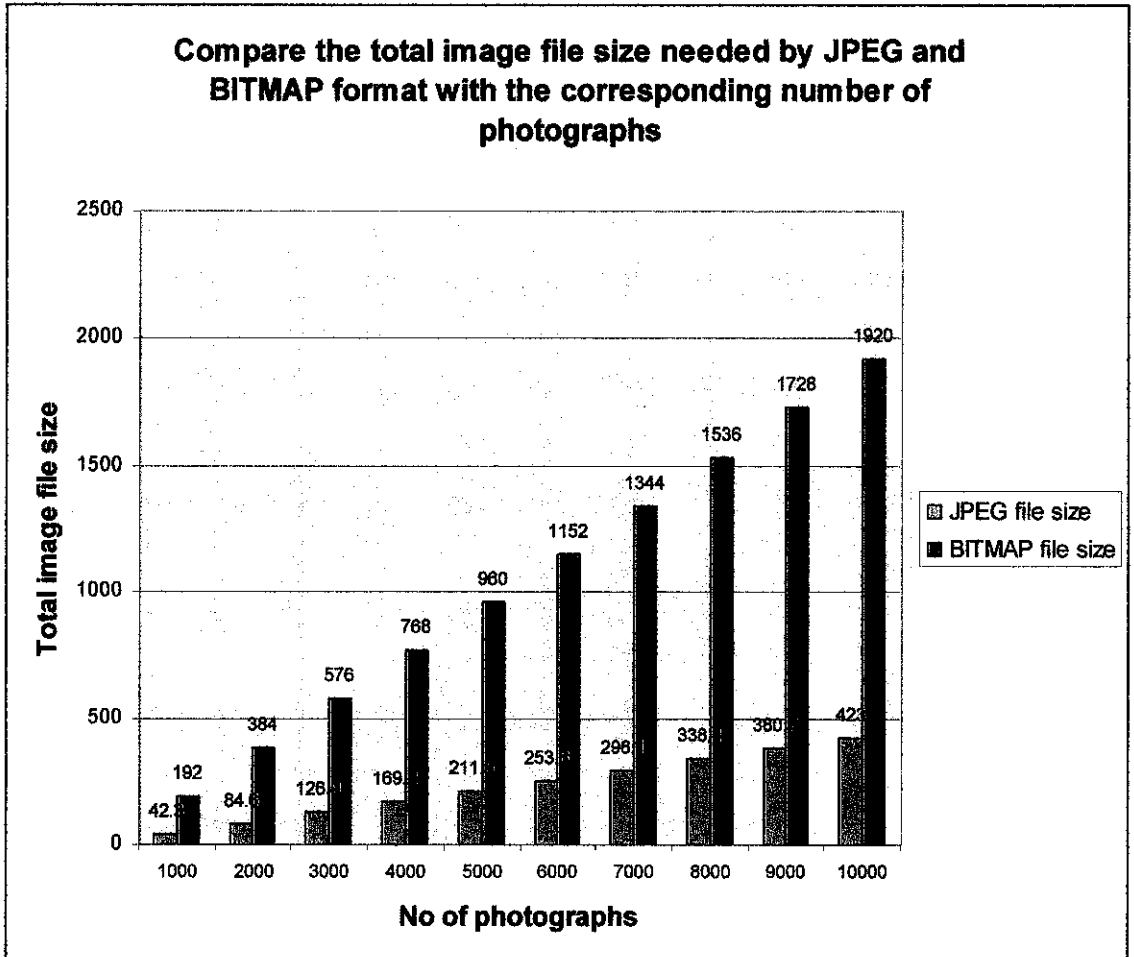


Figure 4.26 Total image file size needed by JPEG and BITMAP format file

As shown in Figure 4.26, there is a drastic difference between the total number of image file size consumed by JPEG and BITMAP format for the same amount of photographs. Figure 4.26 shows that for a total of 1000 photographs, JPEG format will consume 42.3MB (Megabyte) but BITMAP format will consume 192MB which needs almost five times more storage than the JPEG format. The difference in storage consumed by JPEG and BITMAP format files becomes more drastic when more photographs are needed, as shown in Figure 4.26.

Thus it is better if a JPEG file loader can be used in the future so that the amount of memory storage consumed can be reduced. By using image file format that consumed less memory storage, a walkthrough with more footsteps can be implemented, since more photographs can be used with lesser memory storage consumption.

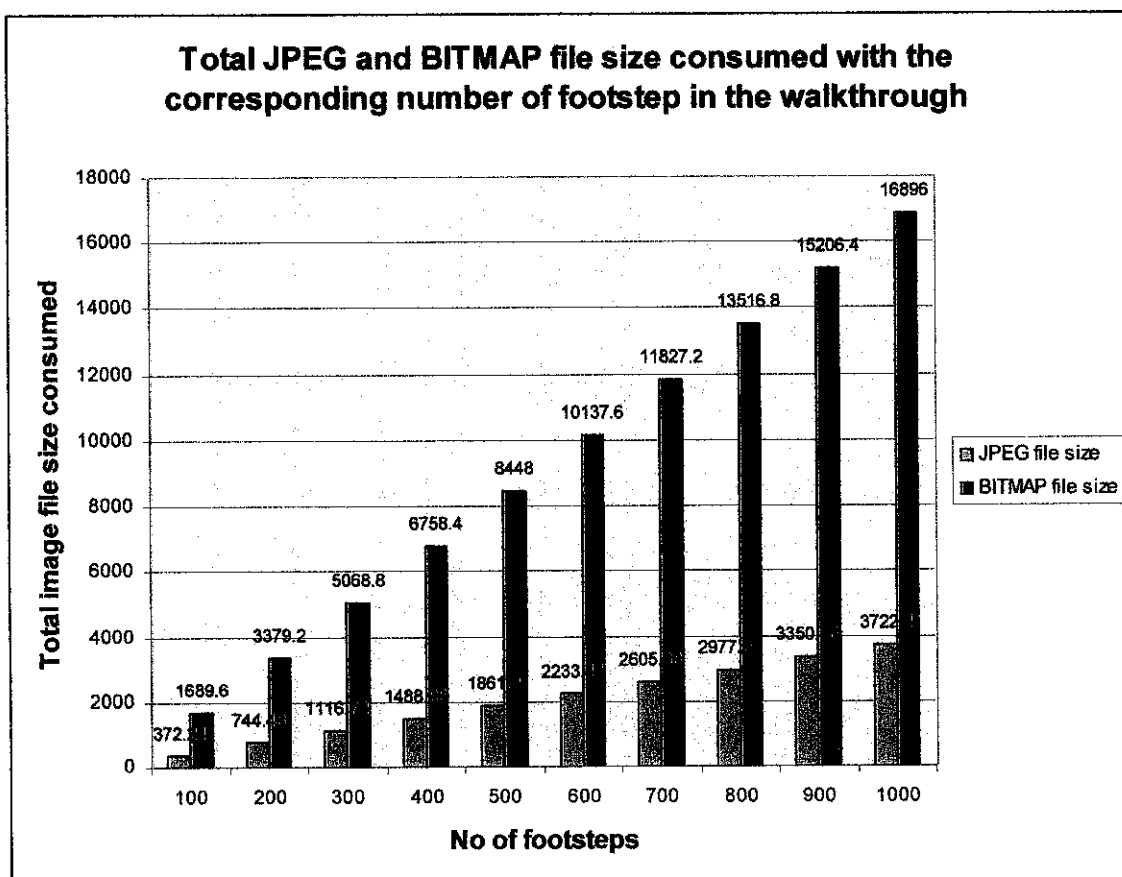


Figure 4.27 JPEG and BITMAP file size consumed with corresponding number of footsteps in the walkthrough

Figure 4.27 shows that to construct a virtual walkthrough with 1000 footsteps, JPEG files will only need 3722.4MB of memory storage where else BITMAP files need a

total of 16896MB of memory storage. Thus it is preferable to use a JPEG file loader in the future to save memory storage. Although there is not much difference in image quality perceived by human eyes; as explained in Section 4.5.1, the big difference in image file size does make a great difference in memory space consumption in the future when more photographs are needed to construct a complete virtual walkthrough for a large environment.

4.6 Speed (frame rate)

The speed of the virtual walkthrough prototype is measured in terms of frames per second. The frame rate measured for the virtual walkthrough prototype constructed is 23 frames per second for a windows display size of 1400x1000. The ideal frame rate used in MPEG for moving video sequence is 60 frames per second. The slow frame rate produced by the prototype constructed might be affected by the large amount of memory storage consumed by the photographs which leads to slow in displaying the virtual scene. However the slow frame rate does not affect the quality of the walkthrough much because the lag is within bearable time limit which is less than one second.

4.7 Smoothness of the virtual walkthrough

The constructed virtual walkthrough prototype is considered smooth. There is very little lag in presenting the virtual scene. Even when there is a lag occurs, the lag in presenting the virtual scene is still within the bearable time limit.

However, due to no specific measurement is used when taking the panoramic photographs, jerkiness occurs during the virtual walkthrough. As shown in Figure 4.28, at line 1 position 3, a smaller number of photographs with bigger degrees of angle are taken; where else at line 1 position 4, a greater number of photographs with smaller degrees of angle are taken. The difference in degrees of the photographs took between 2 subsequent position will affect the smoothness of the virtual walkthrough.

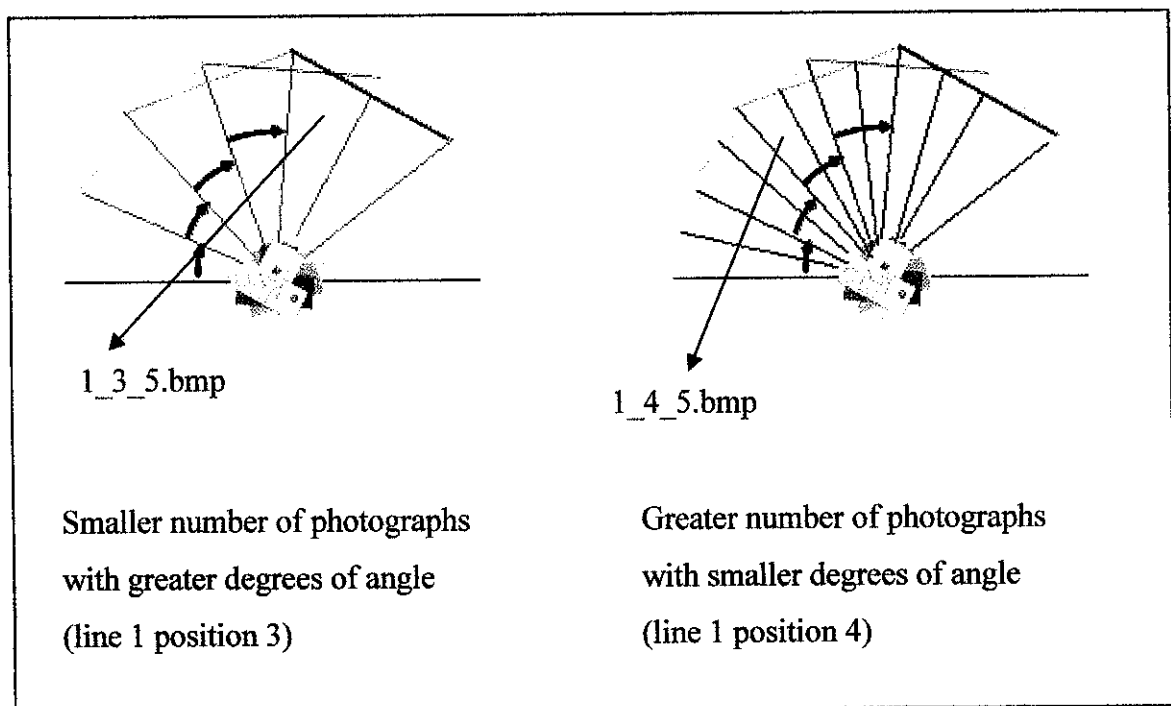


Figure 4.28 Difference in angle when taking photographs

As an example, in Figure 4.29, the viewer is now at location line 1, position number 3 and looking at angle number 5; please refer to Figure 4.28 as well. The virtual scene presented is the photographs took at line 1 position 3 and at angle number 5, which is 1_3_5.bmp. However due to the difference in degrees with the photographs took between position 3 and position 4, when the viewer moves forward to position number

4, the virtual scene will be presenting photographs took at line 1 position 4 and at angle number 5, which is 1_4_5.bmp, please refer to Figure 4.28. The degrees of viewing position change even though the viewer is strictly moving forward without turning left or right. Thus, without proper measurement when taking the panoramic photographs, the smoothness of the virtual walkthrough has been affected.

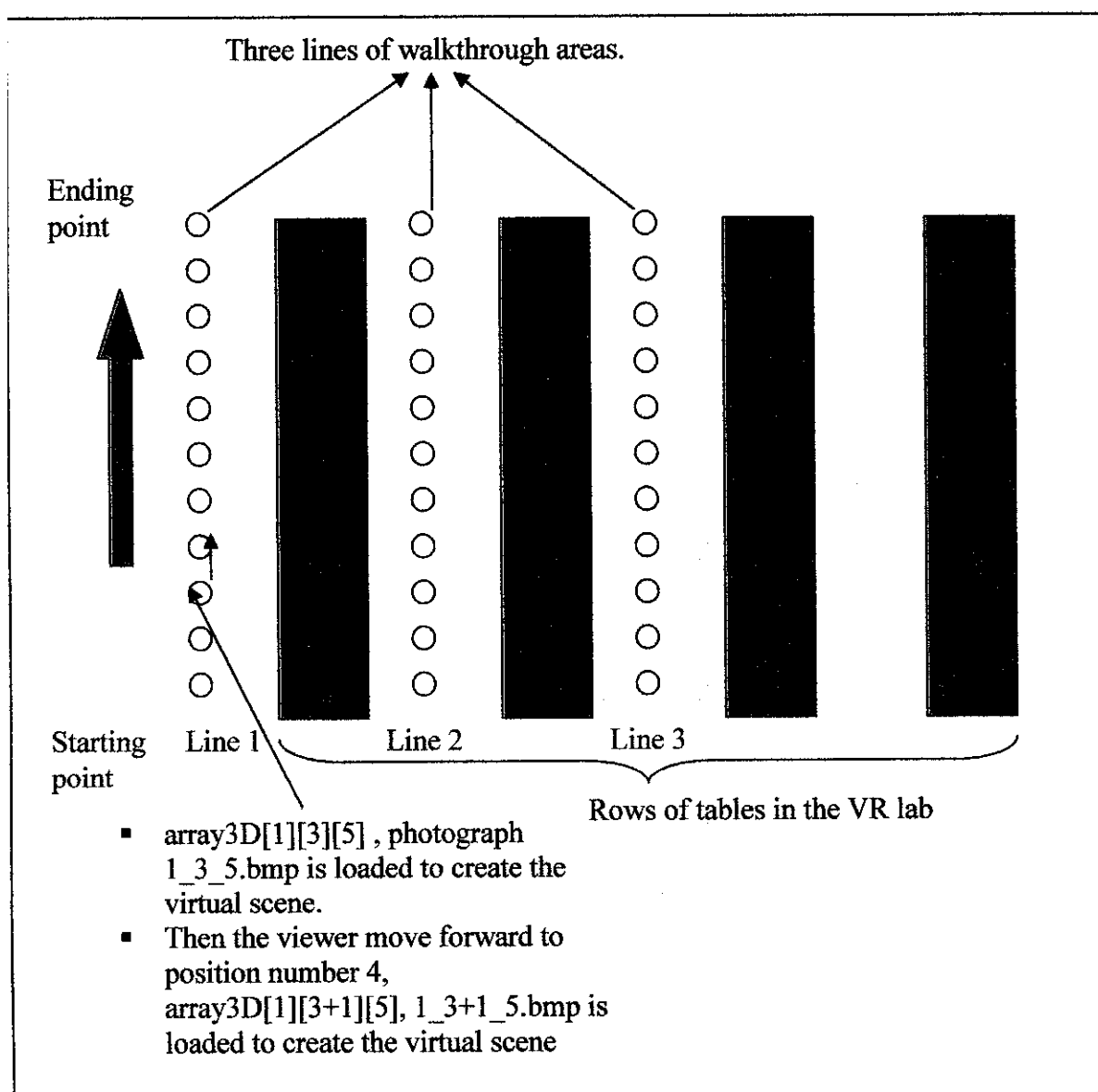


Figure 4.29 Changes in viewing angles during the walkthrough

Tripod is not used when taking the photographs, the photographs are taken based on the photographer's eye height plane, the camera height level change a little when the photographer took every subsequent photograph. Thus, when the viewer moves around in the virtual environment, the height plane of the scenes displayed is not level, making the scenes look a little shaky during the transition from the pervious scene to the current scene. However this uncontrolled change in the camera level which leads to slightly different height level photographs does not reduce the quality of the virtual walkthrough, in fact it makes the virtual walkthrough more realistic. A slight change in viewing level when moving forward in the virtual environment provides a more realistic experience that mimic the walking movement of a human in real life.

4.8 Quality of the virtual scene display

The quality of the virtual scene displayed by using BITMAP photographs is considered good, as shown in the screen captured in Section 4.1. However the displayed BITMAP photographs are not as clear and as realistic as the original JPEG photographs. The colors of certain parts in the BITMAP photographs are not similar to the colors in the JPEG photographs.

As mentioned above the original JPEG photographs are converted into BITMAP photographs due to the format of the texture loader source and header file used support BITMAP format. Table 4.3 shows the difference between the two formats before and after the conversion in order to enable the photographs to be loaded and texture mapped onto the polygon to display the virtual scene.

Table 4.3 Image qualities and file size before and after conversion

Image format	Image width and height (pixels value)	Image file size
JPEG (original format of image captured by using digital camera)	2592 x 1944	482 KB
BITMAP (converted image format)	256 x 256	192 KB

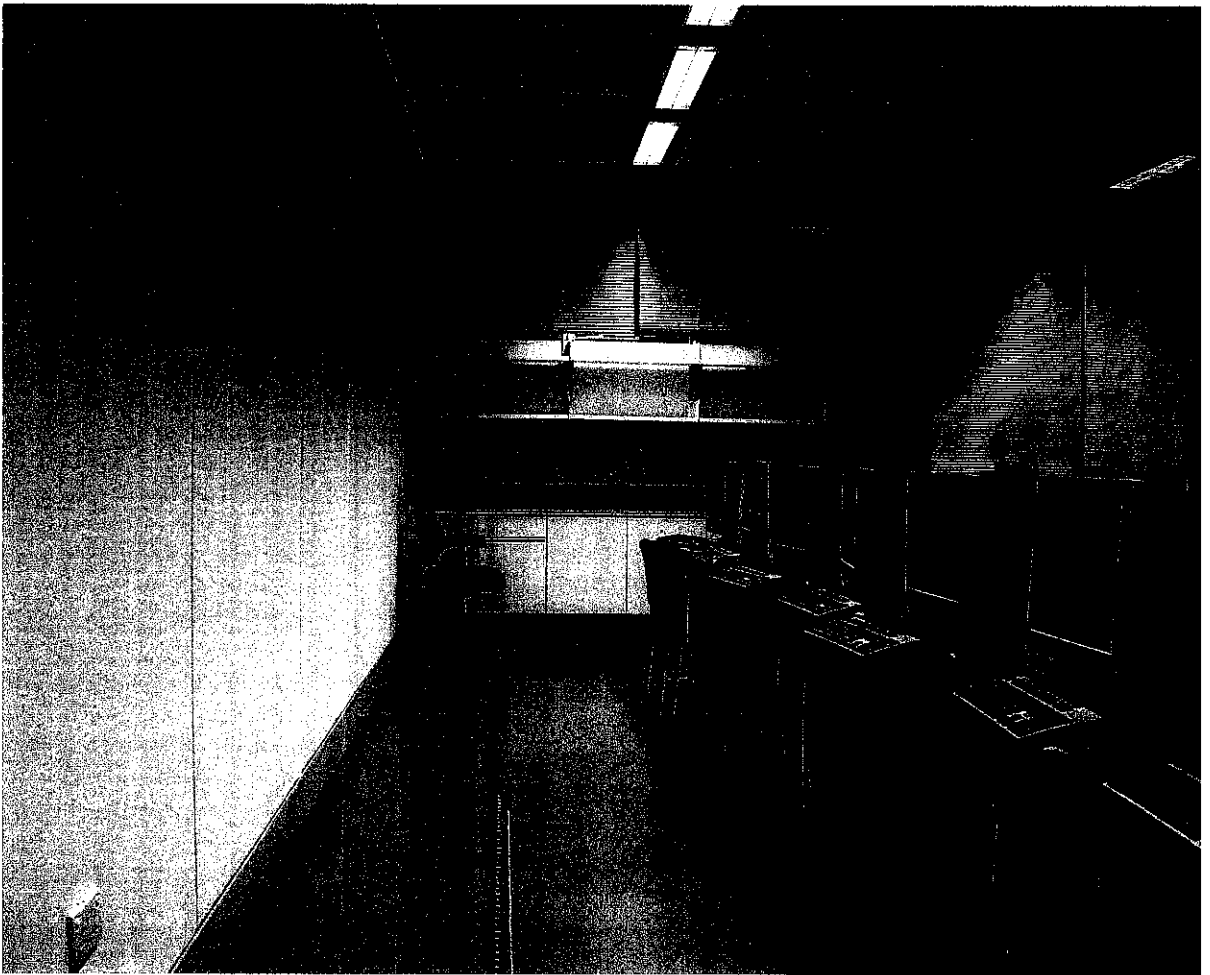


Figure 4.30 Original JPEG image before conversion

Figure 4.30 shows the original JPEG image before conversion while Figure 4.31 shows the BITMAP image generated after conversion. There is noticeable image quality difference between the two. The original JPEG format photograph appears to be clearer and has better resolution due to the higher number of pixels value assigned, while the BITMAP format image looks a little blur and has lower resolution due to a very much lower number of pixels value assigned during conversion.

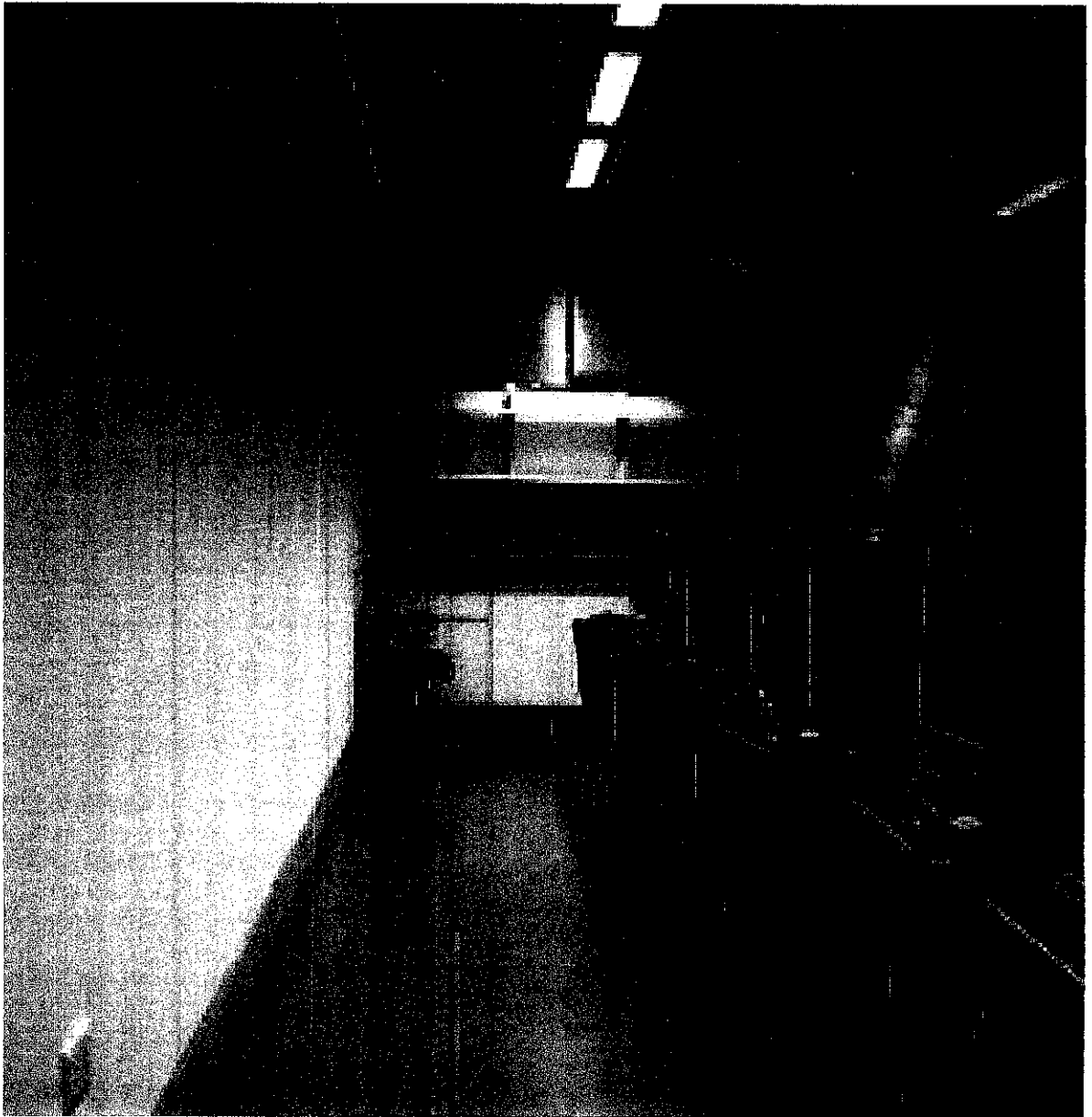


Figure 4.31 BITMAP image after conversion

As a result, it is better to use a JPEG format texture loader source and header files that are able to load image files with greater number of pixels value so that the image quality used for the virtual walkthrough has greater resolution.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

As a conclusion, the virtual walkthrough prototype constructed gives a satisfactory result. This project has surpassed the objectives that have been set initially. After the completion of the initial scope and objectives set for this project, potential areas of improvement are found which lead to the continuation of the research on prefetching and caching mechanism to enhance the virtual walkthrough system. This project has a combination of the best features from the following research papers: *Sea of Images (2001)* by Aliaga, D. G., Funkhouser, T., Yanovsky, D. and Carlbom, I., *Image-based rendering for real-time applications* by Bauer, M and *CyberWalk: A Web-Based Distributed Virtual Walkthrough Environment (2003)* by Chim, J., Lau, R. W. H., Leong, H. V. and Si, A.

With the development of this project, more understanding on IBR technology is gained from the research done throughout the project implementation phase. Through the successful implementation of this prototype, a better and improved approach has been introduced to provide a better alternative to develop virtual walkthrough application in the future.

5.2 Future work

The future work recommended for the current virtual walkthrough prototype will be to enhance the architecture of the current system into a client and server architecture, as shown in Figure 5.1.

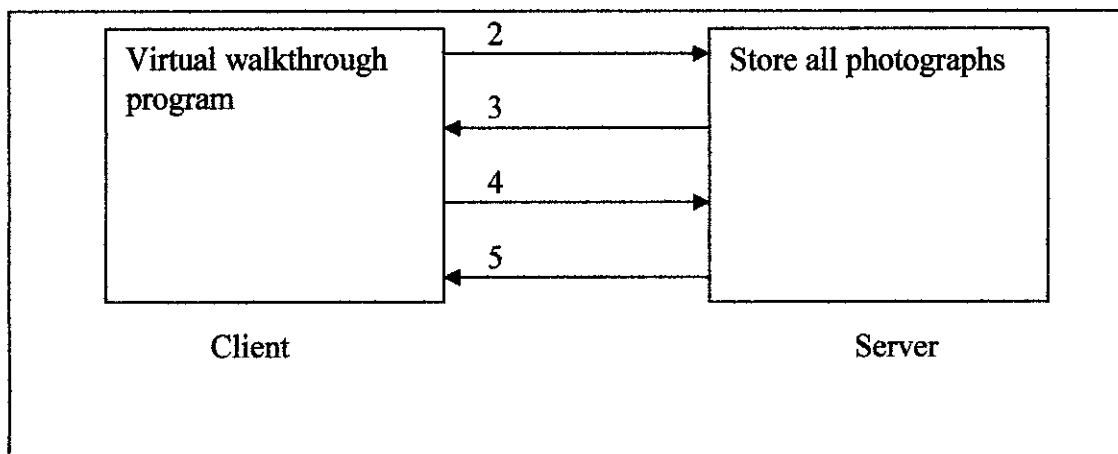


Figure 5.1 Client server architecture

All the photographs needed for the virtual walkthrough can be stored in the server, by doing this the burden of the client can be reduced and also the processing power of the client can be increased to eliminate latency in displaying the virtual scene. The internal processes in running the virtual walkthrough will be as follow:

1. Startup both client and server.
2. Client connects to the server.
3. Server accepts the connection.
4. During the virtual walkthrough, the client sends a message to the server with the name of the photographs that need to be downloaded.
5. Server received the message and sends the photographs to the client.

As discussed in Section 4.3, the Mean Prediction Scheme used cannot adapt quickly to the changes of viewer's moving pattern. It is recommended that Window Prediction Scheme and Exponentially Weighted Moving Average (EWMA) Prediction Scheme are also implemented to the system to test the effectiveness of these two schemes and compare with the Mean Prediction Scheme to find out which prediction scheme can predict the next moving vector the best during the virtual walkthrough. With better and more accurate prediction of the viewer's next moving vector, it is for sure the right photographs will be prefetch to the client to reduce latency in presenting the virtual scene.

As mentioned in Section 4.4, the caching mechanism implemented does not have update mechanism to update the cache table. It is suggested that an update mechanism is implemented to frequently update the cache table every time the viewer move to a new location so that the viewer will never move out of the boundary of the caching area.

A history of the viewer's recent previous locations should also be kept track to cache the photographs at the client, if the viewer move back to the recent previous location, the photographs are already available at the client.

Other than prefetching and caching mechanism, another suggestion is to implement MPEG compression and image interpolation into the current system to compress the acquired photographs in a multiresolution hierarchy to enable the massive amount of captured photographs to be accessed efficiently to reconstruct the novel images during the virtual walkthrough. On the other hand, in order to compress the photographs more, it is suggested that forward prediction mechanism is implemented in the current system, by using forward prediction, some of the photographs in the server can be eliminated by predicting the data in the photographs from the I frame and P frames. Thus more compression can be gained and more memory space is saved. Another way to reduce

the memory space consumption is to use a JPEG file loader to load the images rather than BITMAP loader. By using JPEG loader, JPEG image which has much smaller image size can be used as the texture, replacing the big size BITMAP image, which then leads to the creation of a virtual walkthrough for greater area with greater number of footsteps.

REFERENCES

Mitchell, J. L., Pennebaker, W. B., Fogg, C. E., LeGall, D. J. (2001). *MPEG Video Compression Standard*. Chapman & Hall.

Aliaga, D. G., Funkhouser, T., Yanovsky, D. & Carlbom. I. *Sea of Images*. IEEE. pp.331-338.

Bauer, M. *Image-based rendering for real-time applications*. Media Technology and Design. University of Applied Sciences Hagenberg Austria. IEEE

Retrieved *MPEG video compression technique* from <http://rnvs.informatik.tu-chemnitz.de/~jan/MPEG/HTML/mpeg_tech.html> on May, 2006

Retrieved Chereches G., Ramani K., R. M., Wala M. (2003). *Discrete cosine transform* from <<http://vlsi1.engr.utk.edu/~gabi/552/dct/report/intro.html>> on May, 2006

Chim, J., Lau, R. W. H., Leong, H. V., and Si, A. (2003). *CyberWalk: A Web-Based Distributed Virtual Walkthrough Environment*. IEEE Vol. 5, pp.503-515.



APPENDICES

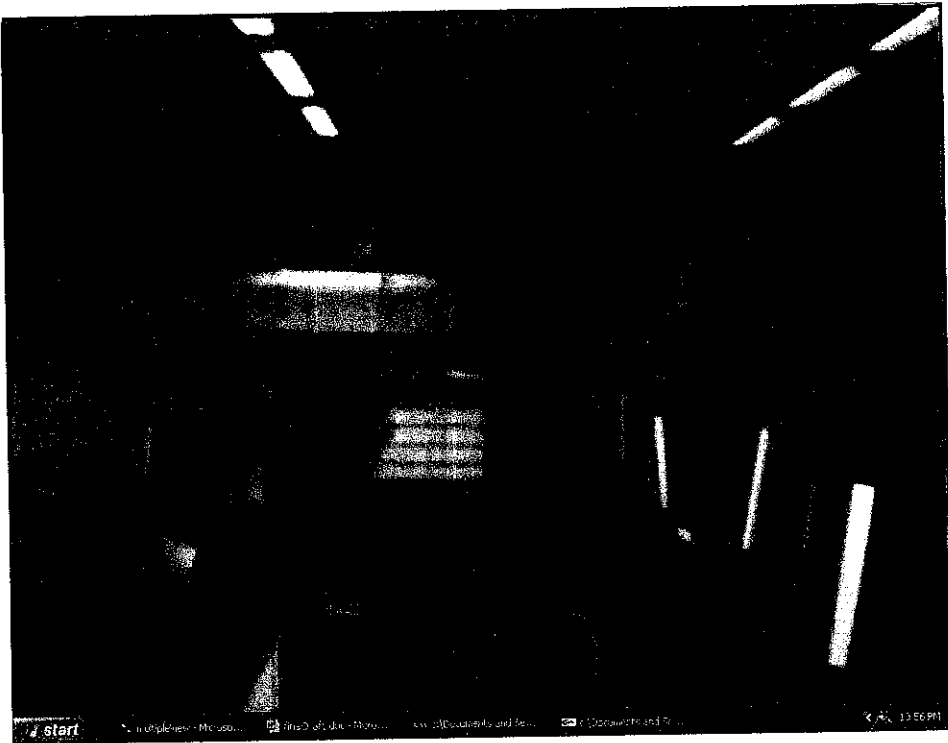
- I. Moving forward and backward in the virtual environment (screen captured)
- II. Making a 180 degrees turn at the fixed spot in the virtual environment (screen captured)

I. Moving forward and backward in the virtual environment (screen captured)

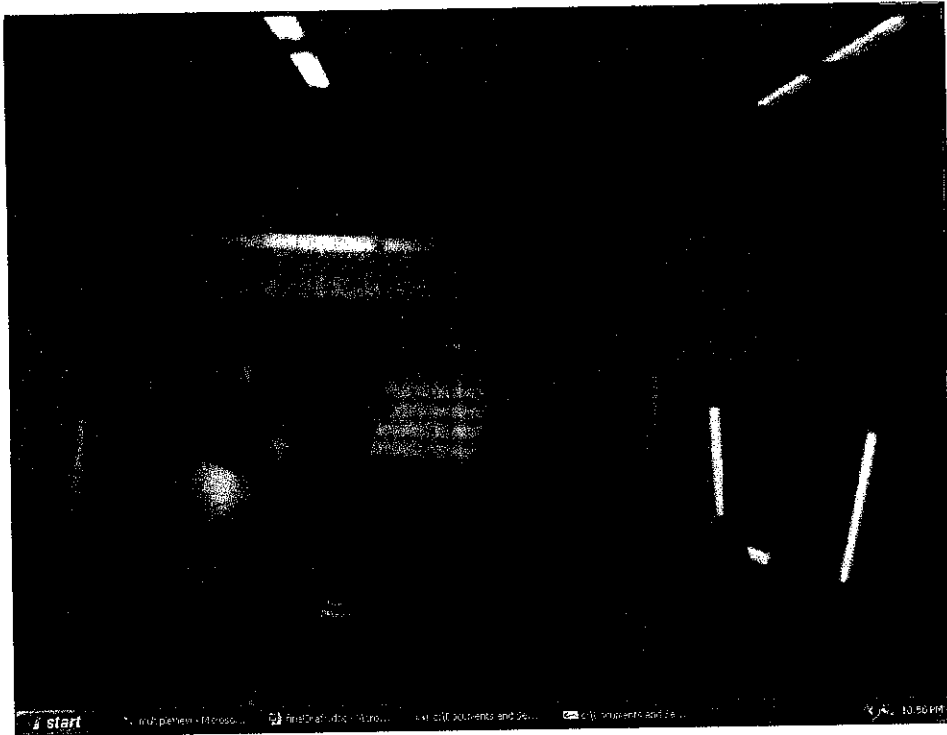
The twelve screen shots below in Appendix I show the change in the virtual scene when the viewer moves forward starting from the location in the screen shot below, which is located at the second pathway.



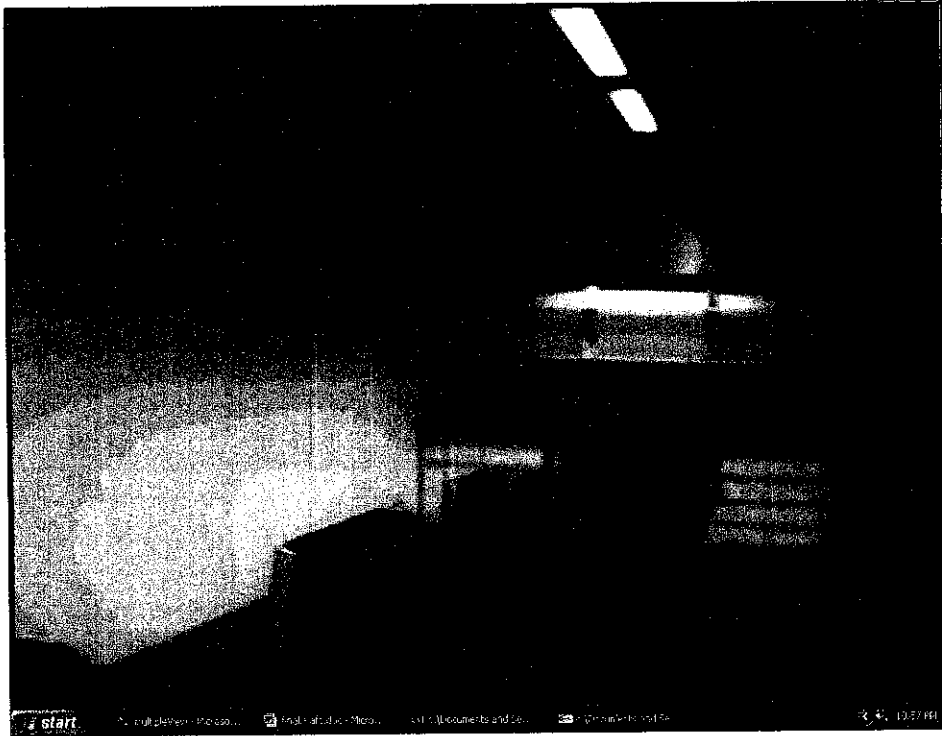
1st step



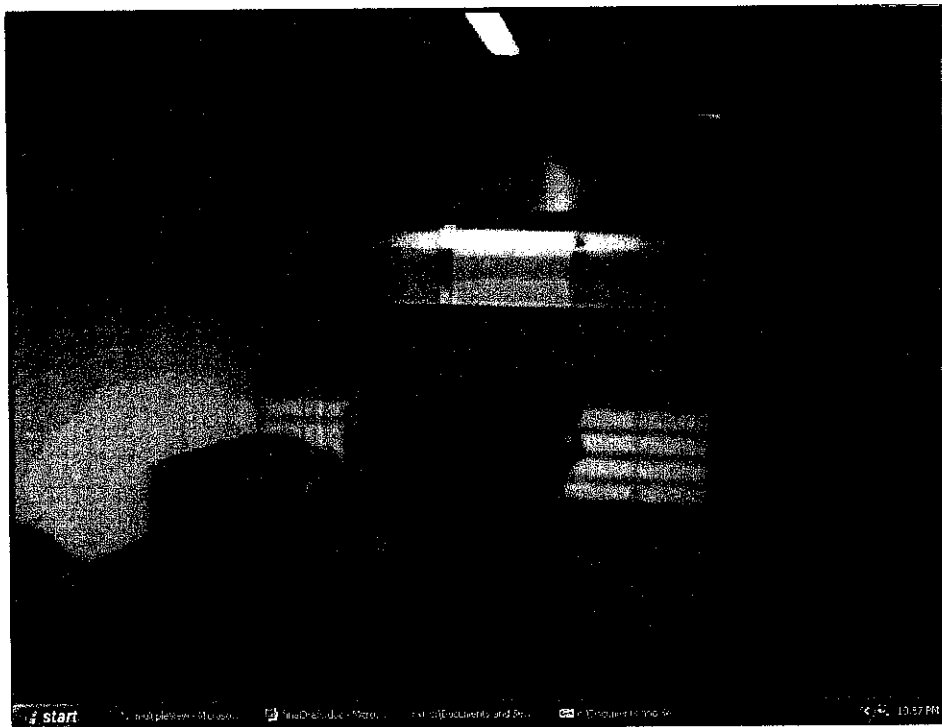
2nd step



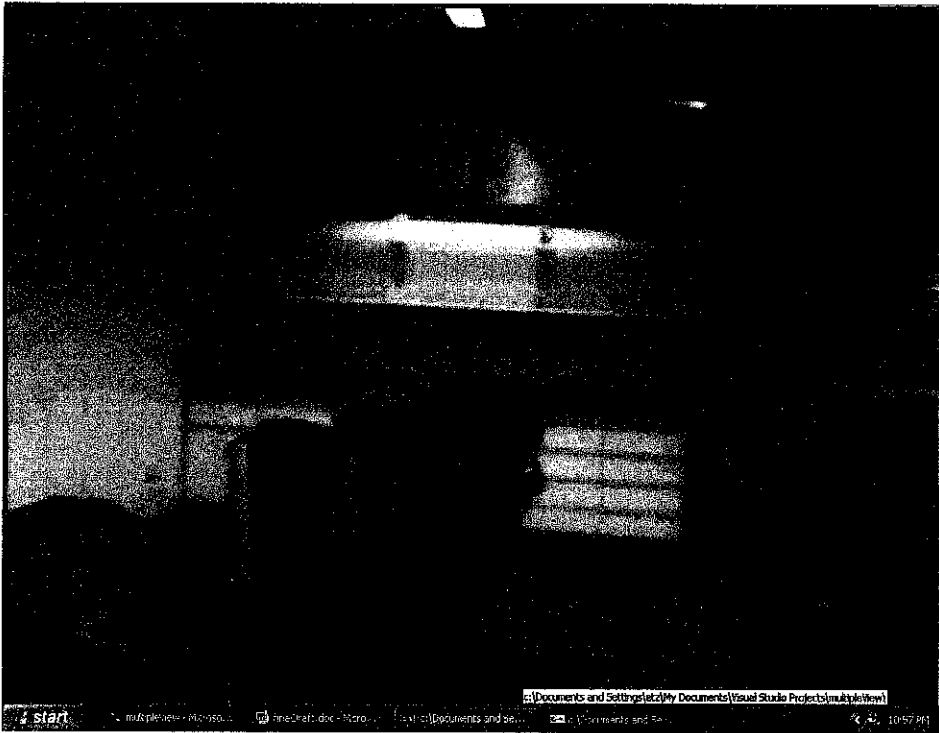
3rd step



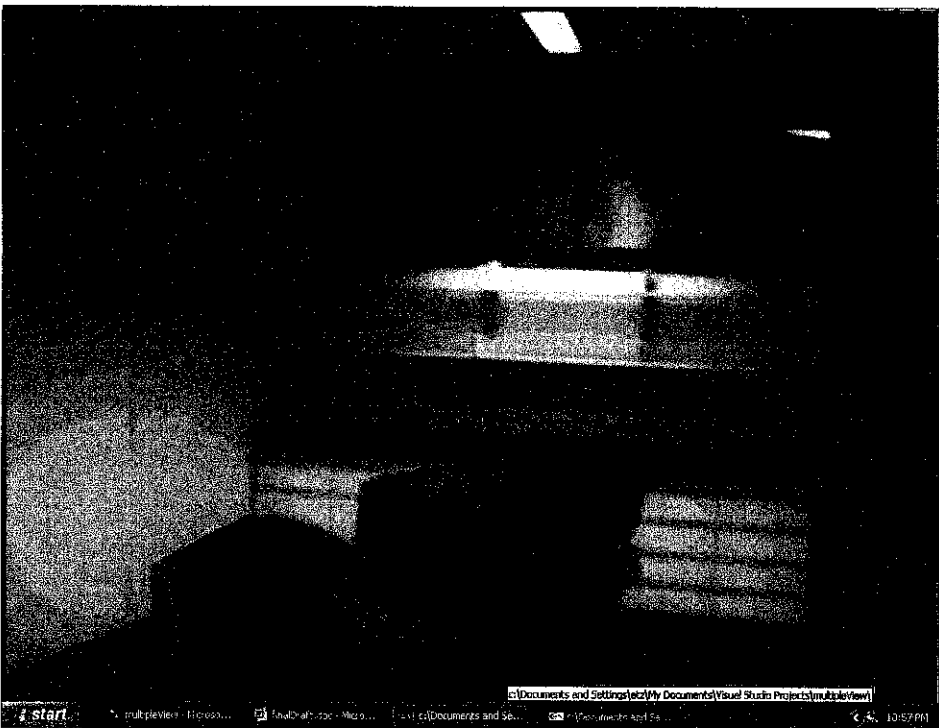
4th step



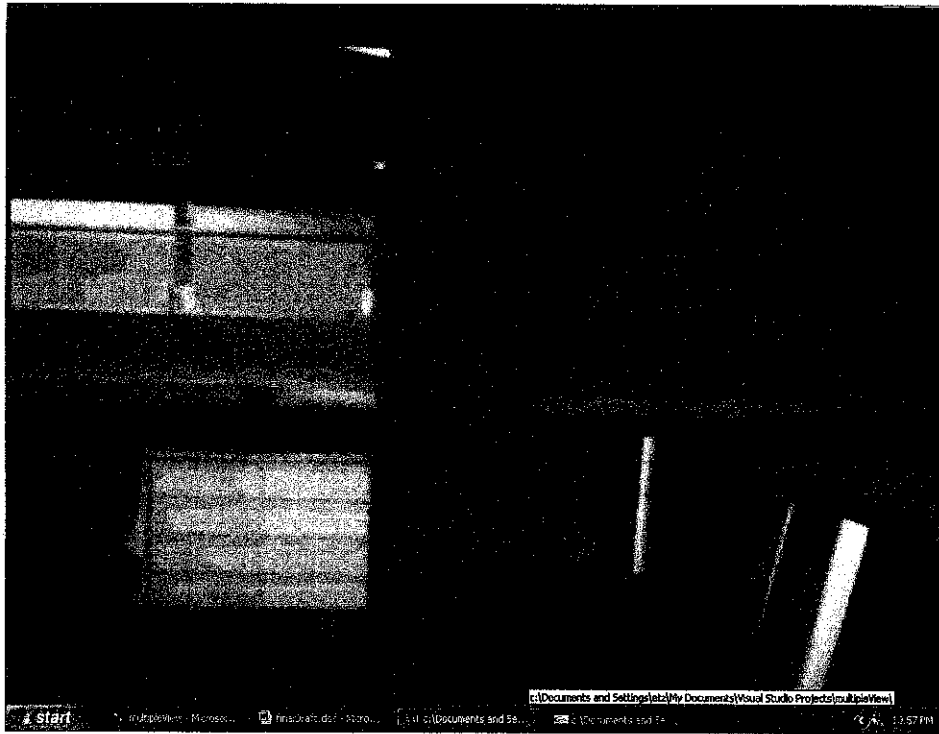
5th step



6th step



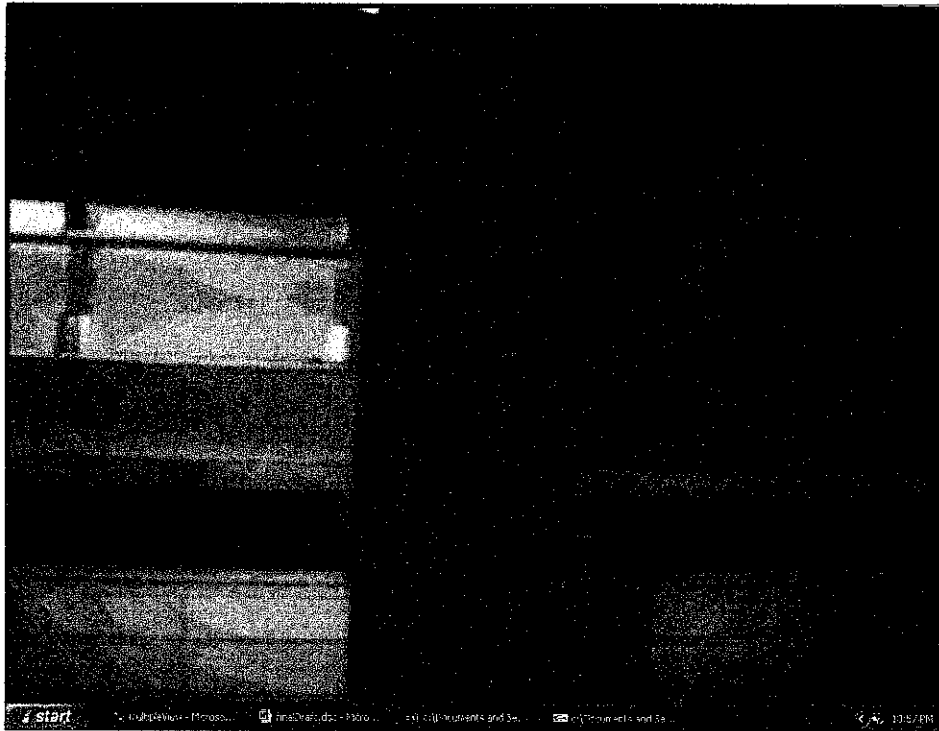
7th step



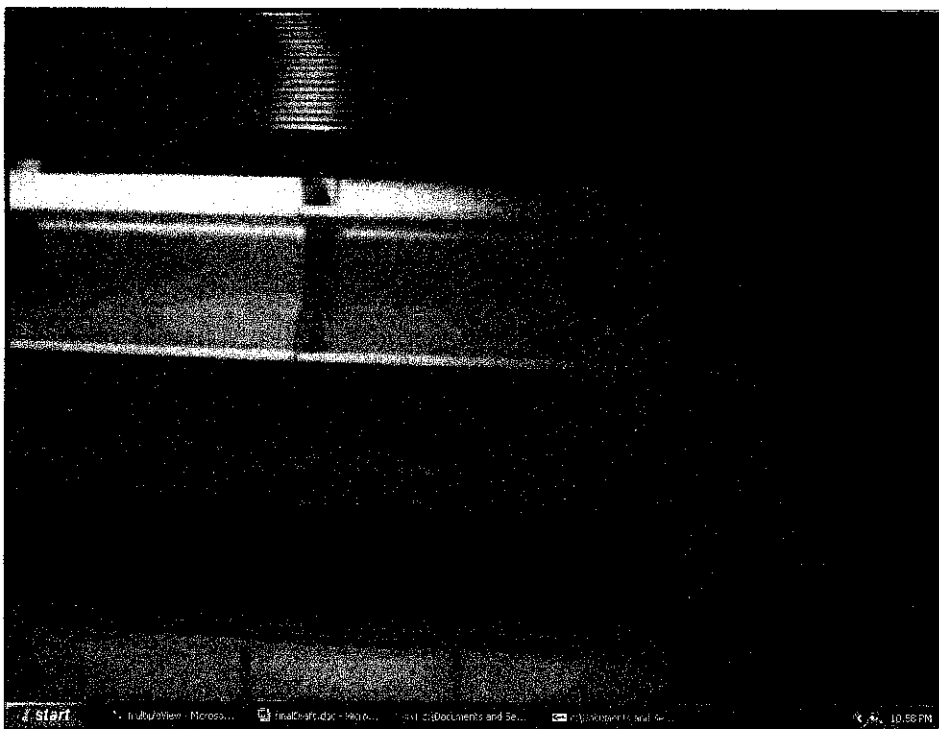
8th step



9th step



10th step



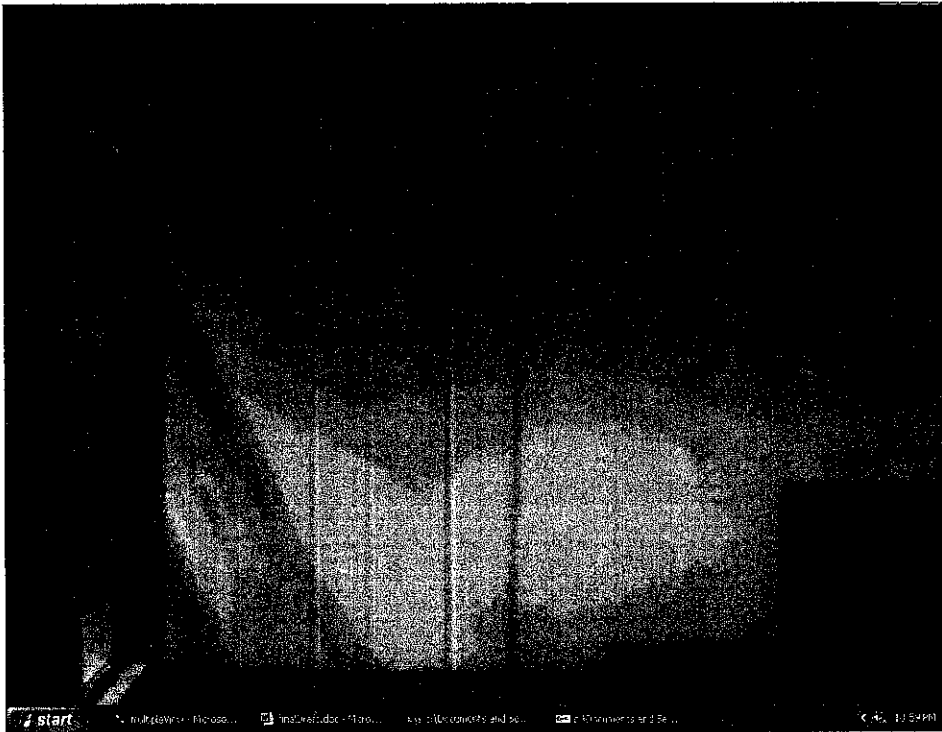
11th step



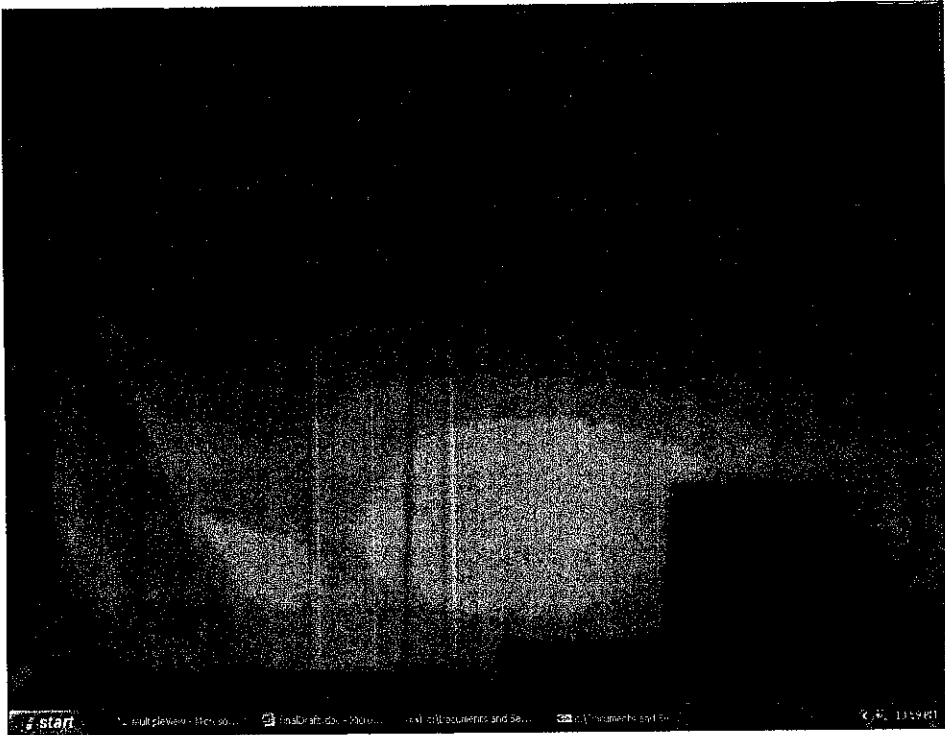
12th step

II. Making a 180 degrees turn at a fixed spot
in the virtual environment (screen
captured)

The forty screen shots below show the change of virtual scene when the viewer turn from left to right at a fixed spot to have a 180 degrees view of the surrounding



1st turn



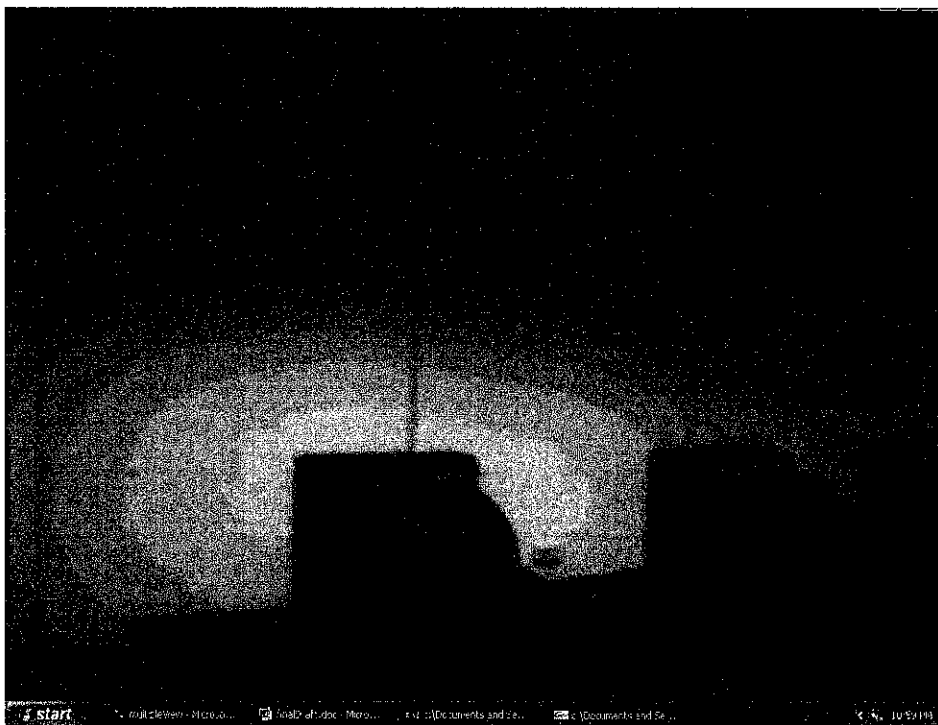
2nd turn



3rd turn



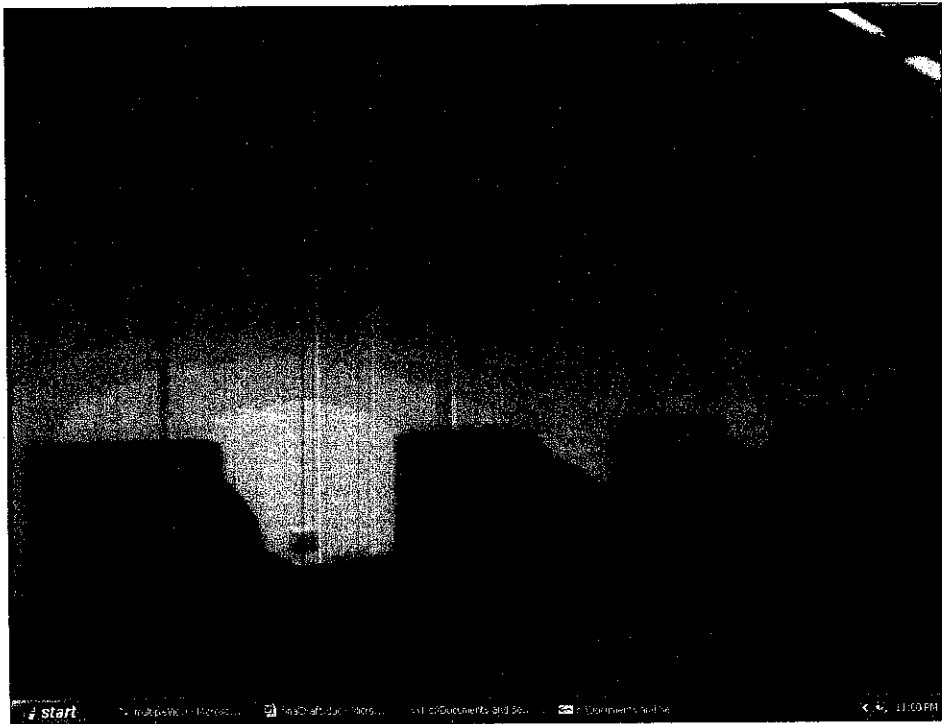
4th turn



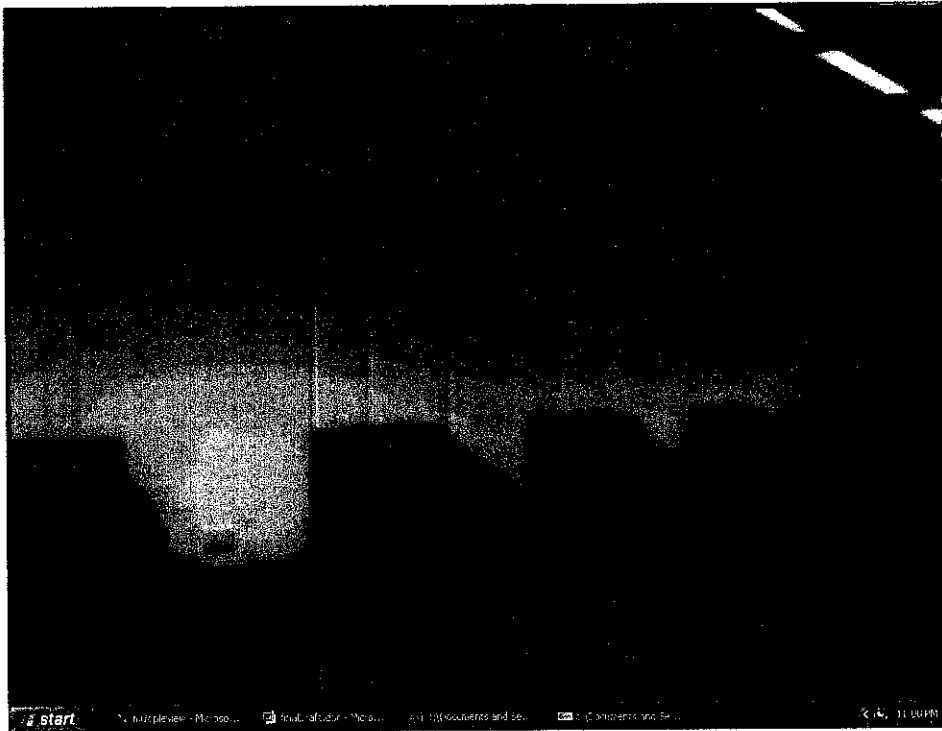
5th turn



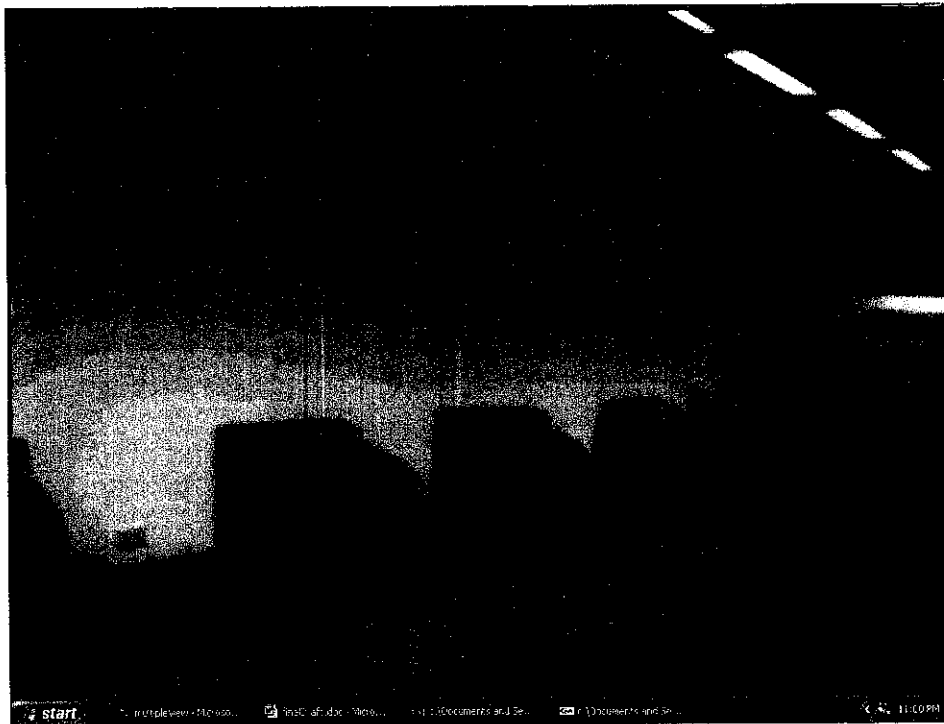
6th turn



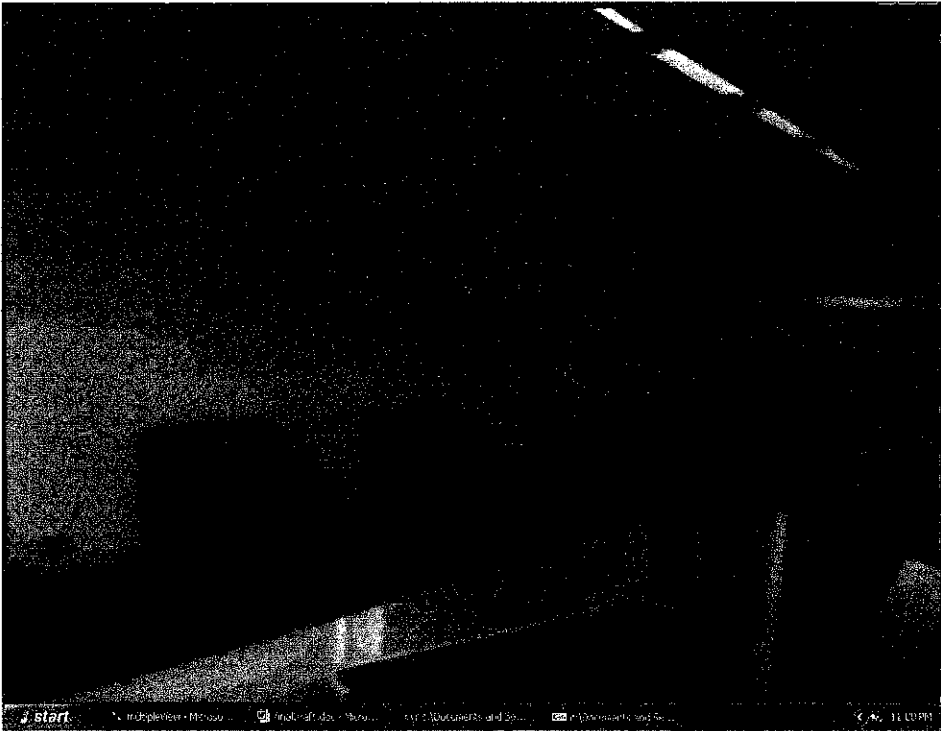
7th turn



8th turn



9th turn



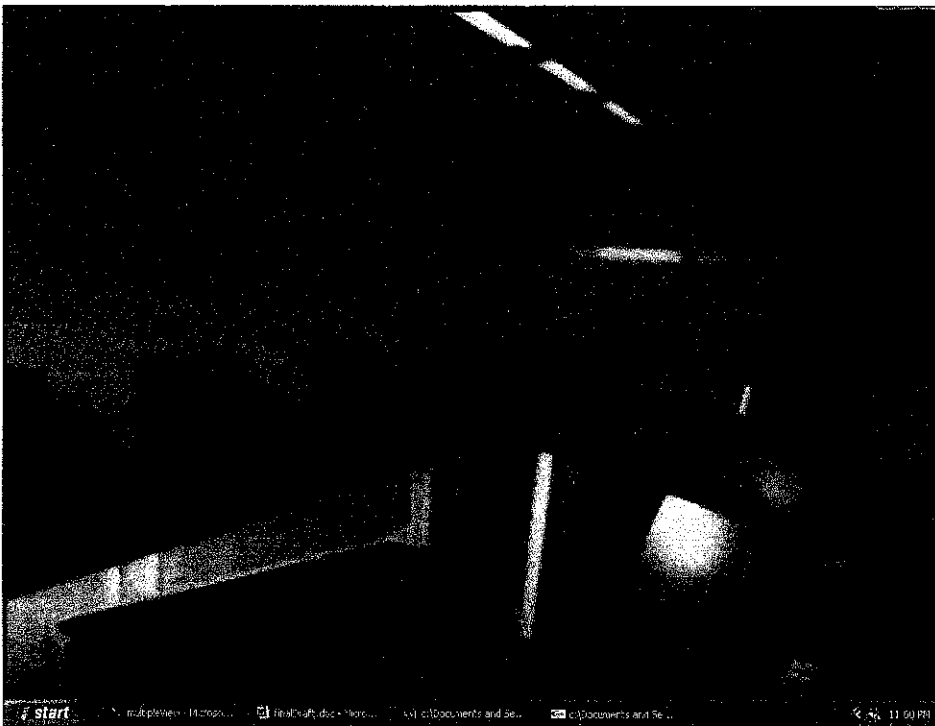
10th turn



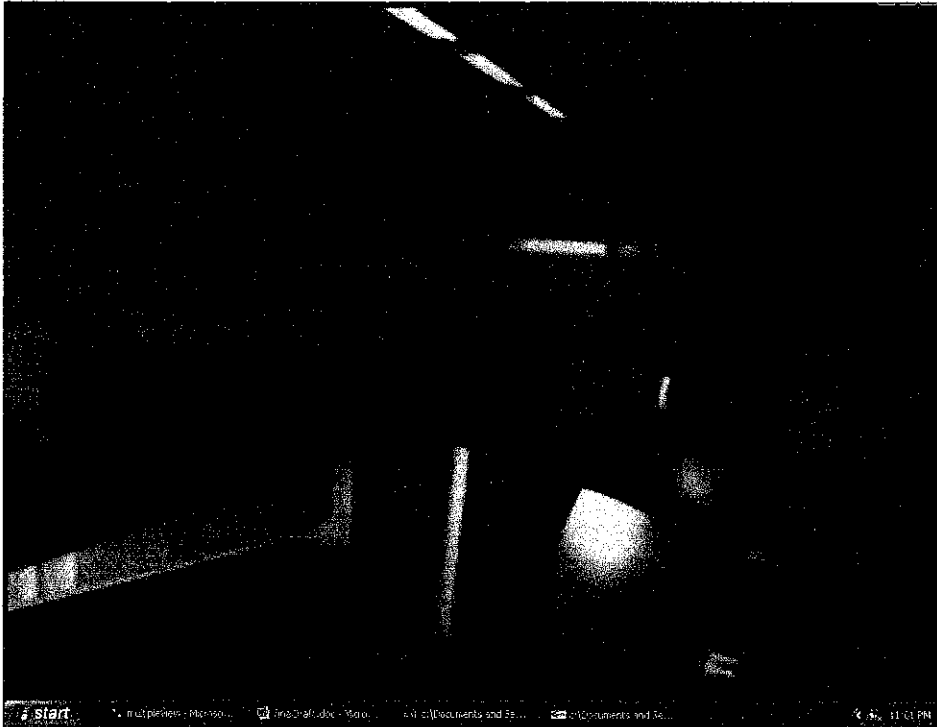
11th turn



12th turn



13th turn



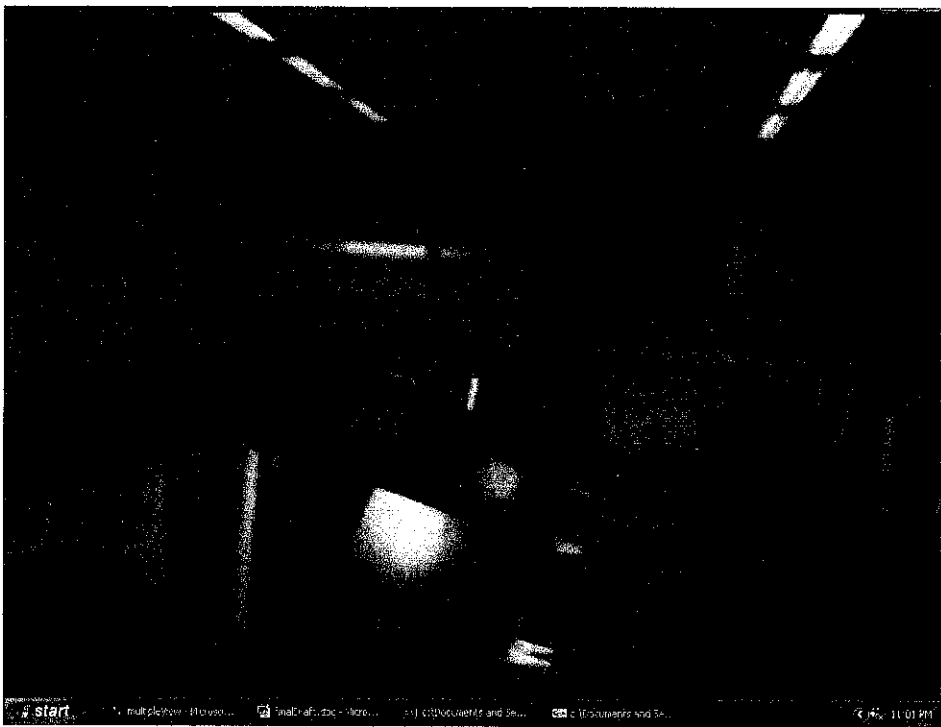
14th turn



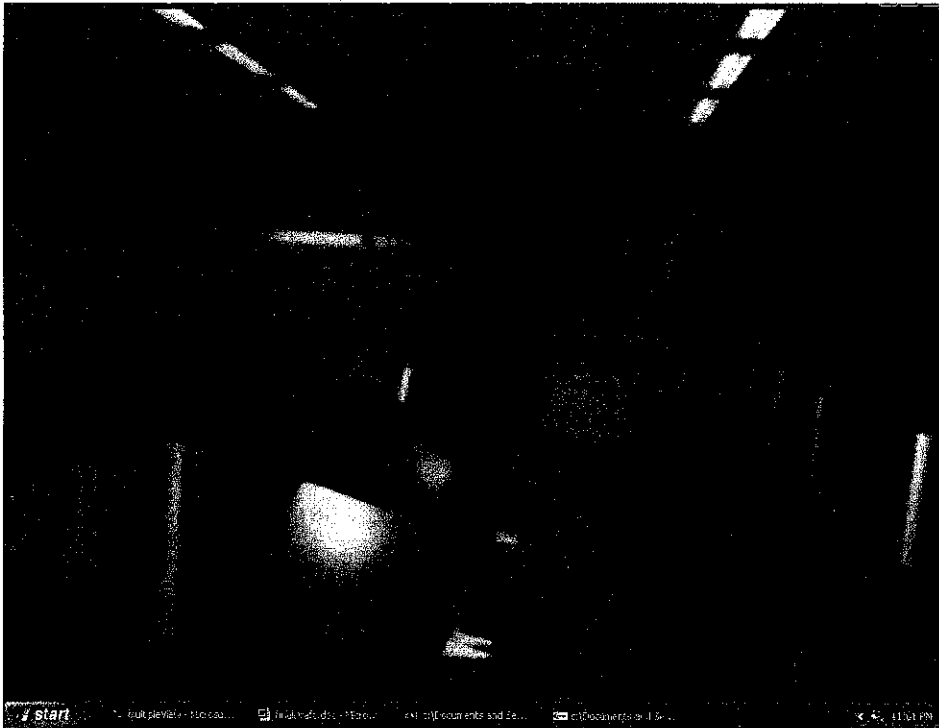
15th turn



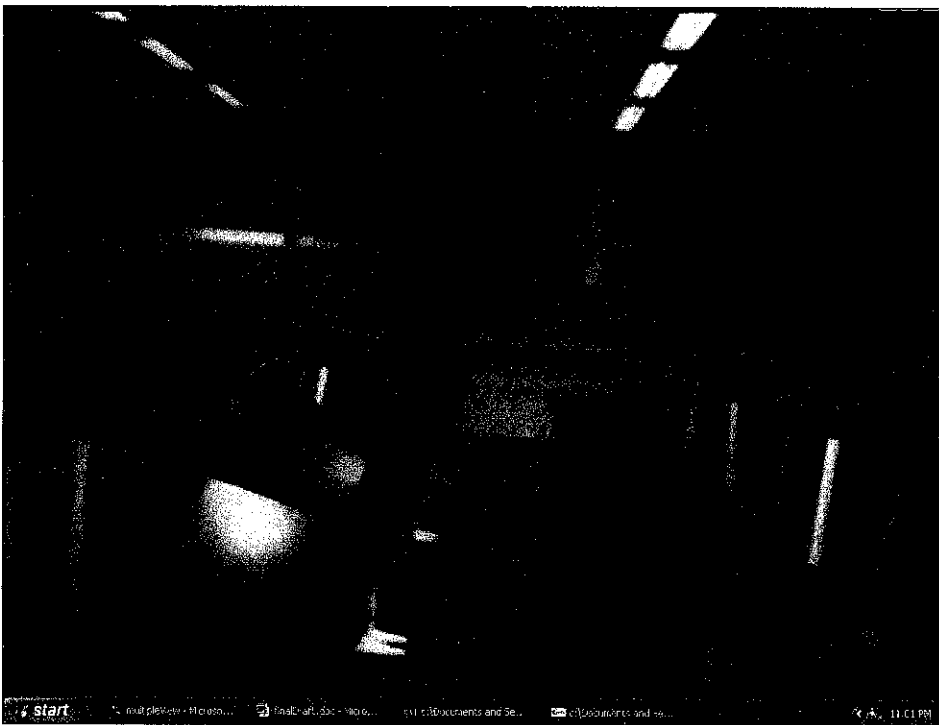
16th turn



17th turn



18th turn



19th turn



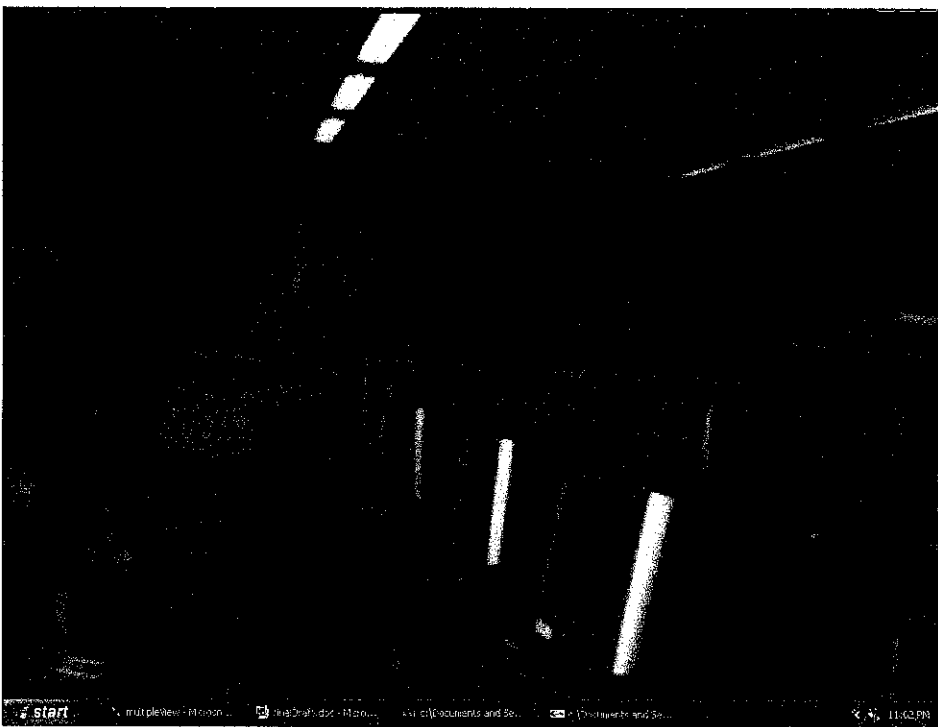
20th turn



21st turn



22nd turn



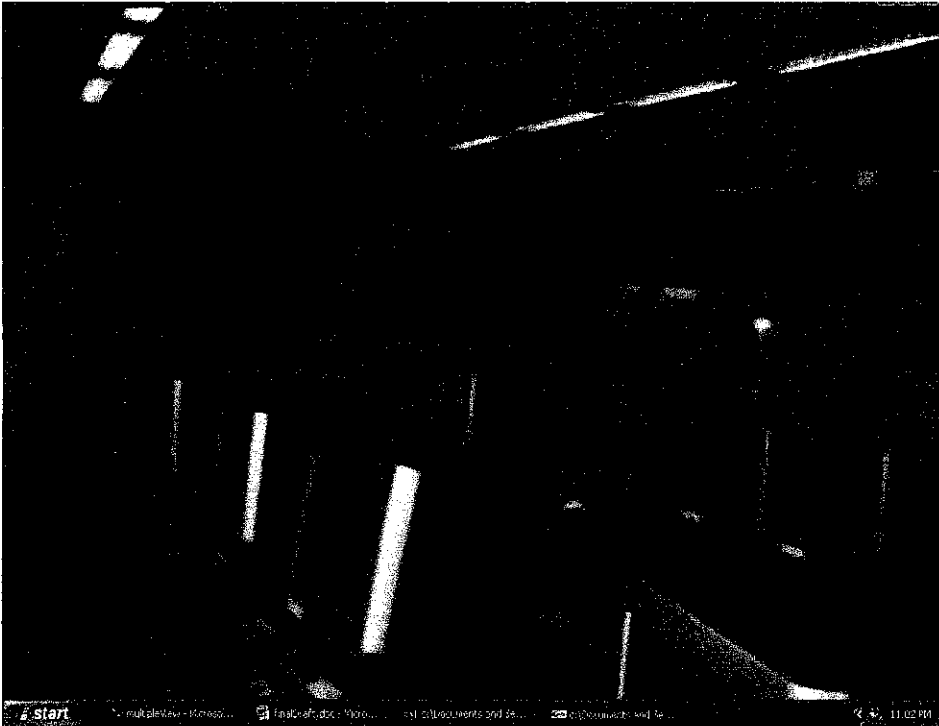
23rd turn



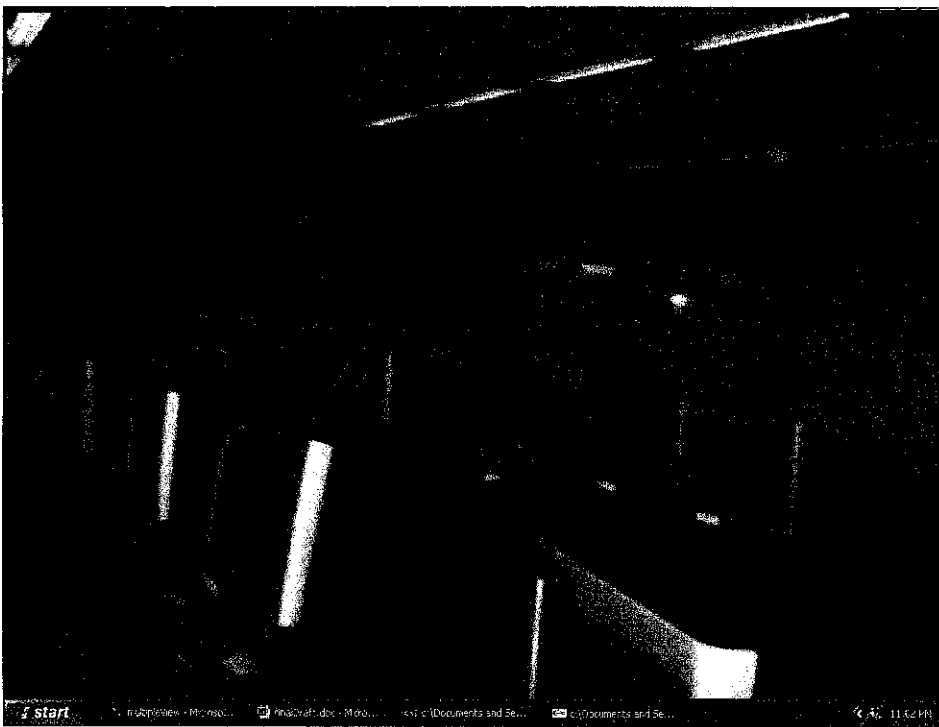
24th turn



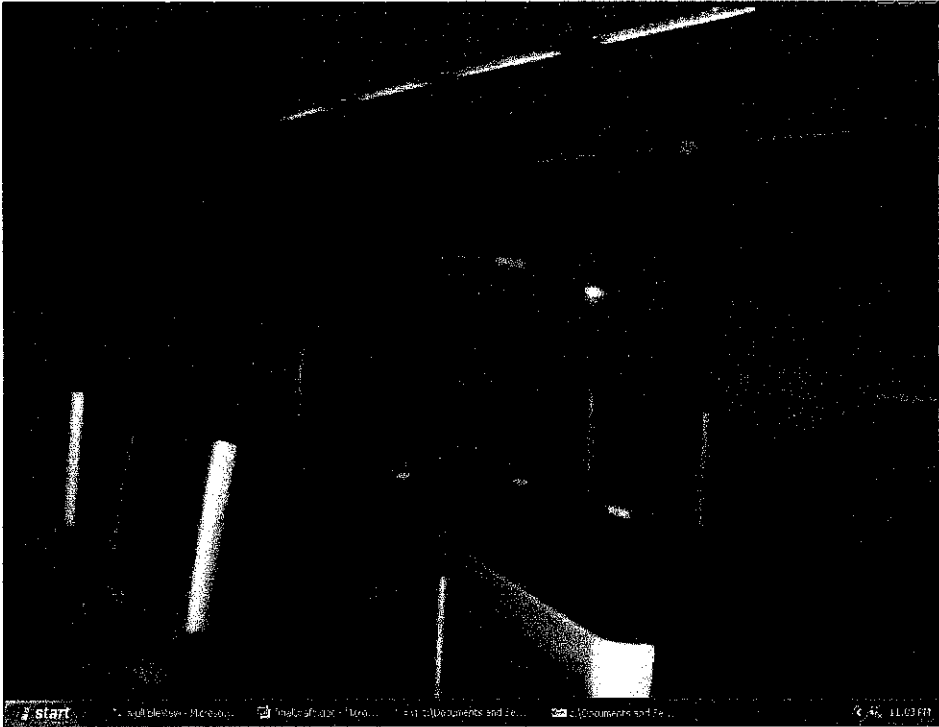
25th turn



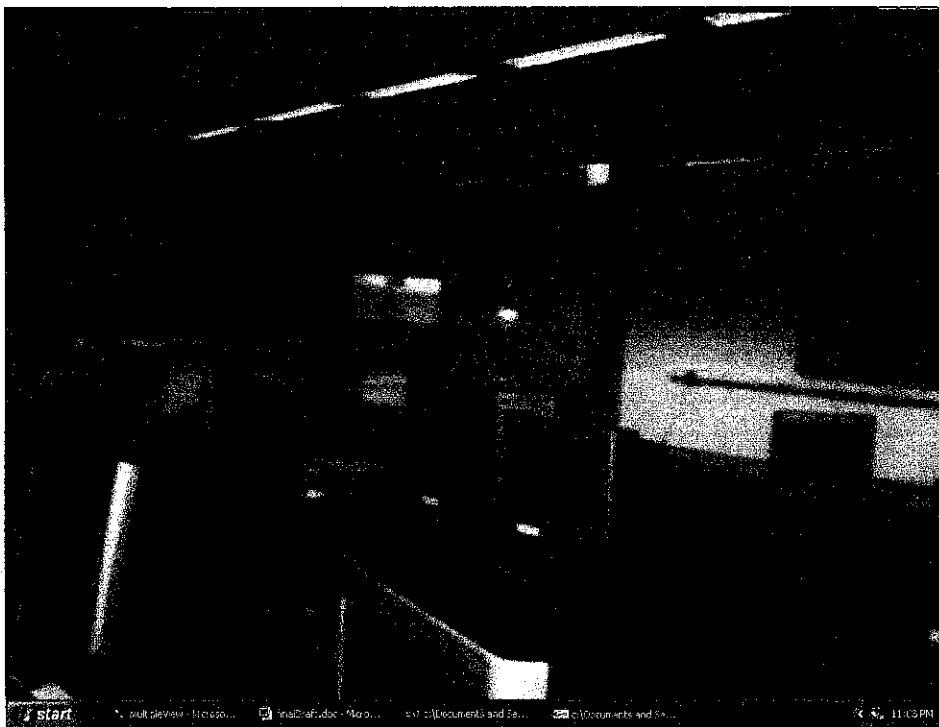
26th turn



27th turn



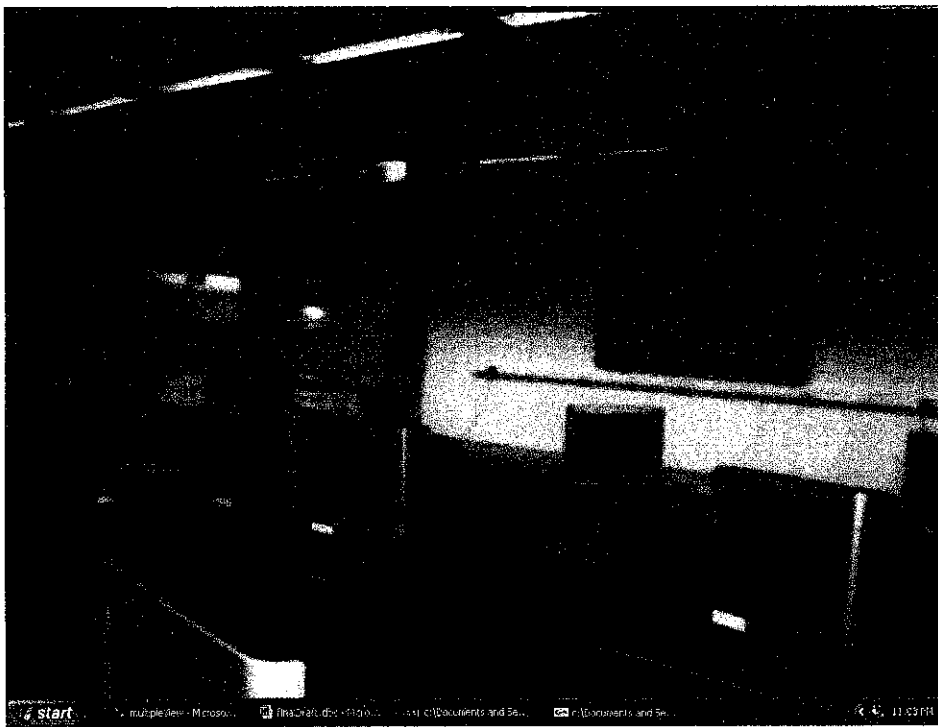
28th turn



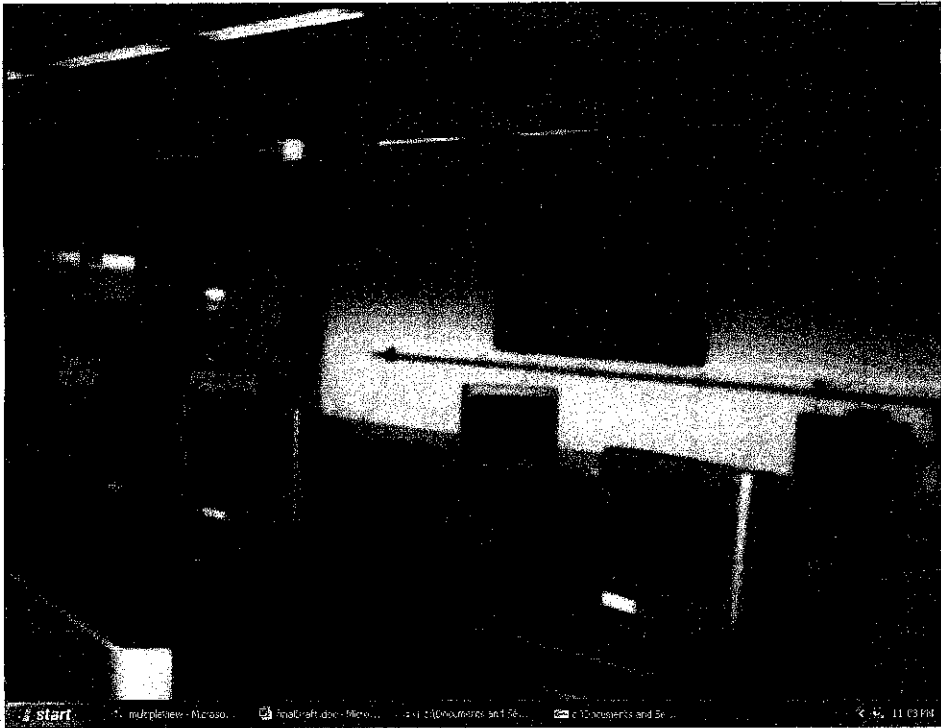
29th turn



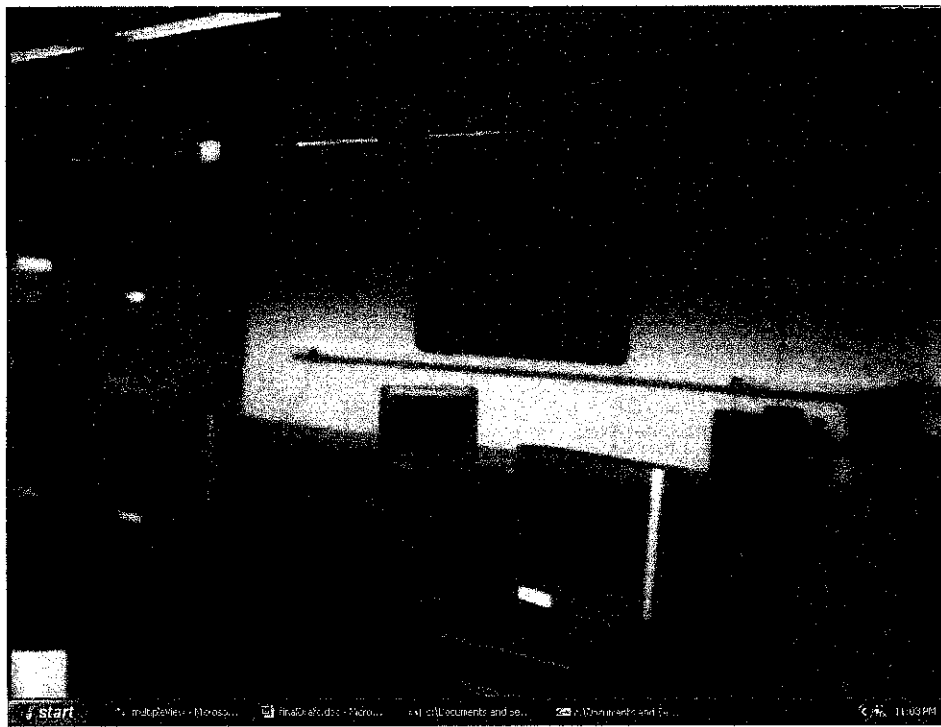
30th turn



31st turn



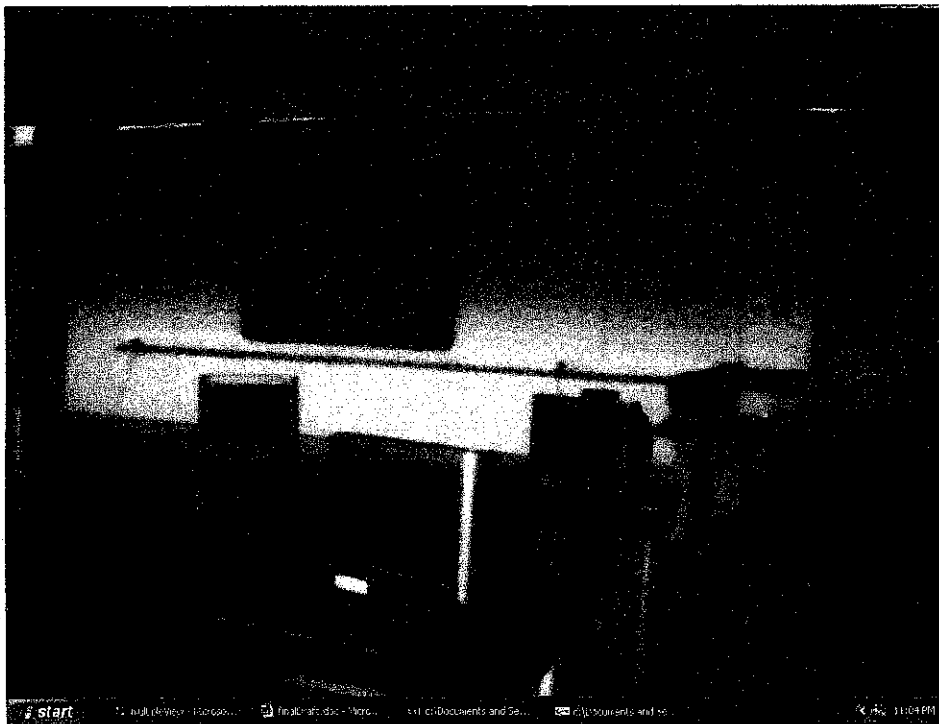
32nd turn



33rd turn



34th turn



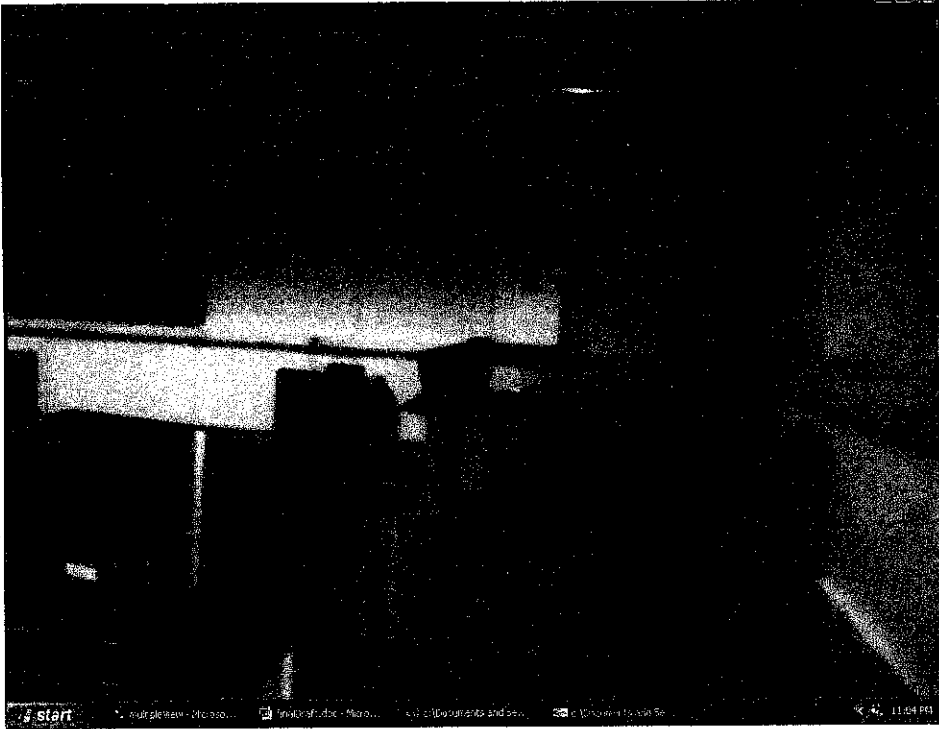
35th turn



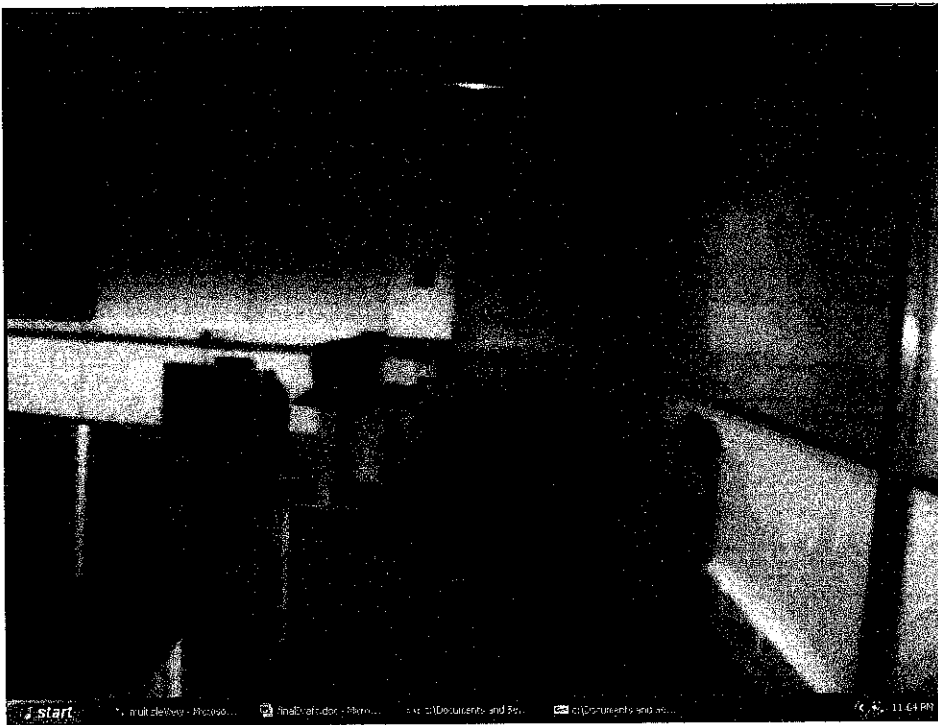
36th turn



37th turn



38th turn



39th turn



40th turn