GENERATION OF 3-D SYNTHETIC AUTOSTEREOSCOPIC INTEGRAL IMAGES USING COMPUTER SIMULATED CAMERAS

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

By

Shafik Salih

Electronic and Computer Engineering
School of Engineering and Design

Brunel University

March 2014

Abstract

Production of artificial Three-Dimension (3-D) images was the aim of many researches over hundreds of years. 3-D images are the images that create sense of depth when viewing them. 3-D images are closer to the real world scenes than 2-D images due to the 3-D effect or the sense of depth the 3-D images provide. Sense of depth can be caused by binocular cues including convergence and parallax. Convergence is created by the difference between the angles of the left eye and the right eye viewing axes. Parallax is the effect of viewing with one eye a view of the scene that is inherently shifted to the view seen by the other eye. Several techniques have targeted the creation of 3-D images with the mentioned cues. The technique is preferred when it is able to create 3-D images so that the viewer can view these images without wearing special glasses and the occurrence of viewer fatigue. Integral photography that was invented in 1908 is able to meet the previous requirements. Based on integral photography, several techniques, research and studies have been published.

The purposes of this thesis include the computer simulation of flexible integral photography systems, the computer generation of good quality 3-D static and animated integral images using the simulated systems, optimising the generation process to be more accurate, less expensive, more effective, and faster, and producing a portable specialist software tool to achieve these targets. New techniques and algorithms are needed to meet these purposes.

A literature survey was carried out about the closest researches and studies to the subject of computer-generated integral images; these were compared with the new techniques introduced in this study to prove the advantages and the necessity of these new techniques. The closest technique to the suggested techniques was implemented using more developed tools to compare the quality of the resulting integral images with the targeted integral images that are going to be produced using the tools and algorithms proposed in this thesis. A method to simulate an imaging system and produce integral images based on the new technique of dividing the view volume of the scene was introduced, explained, proved, and implemented with a program designed for this purpose. To optimise the processing time and the image quality, the previous method is developed, new features are added to the resulting integral images, and better performance was achieved by introducing the method of

Displacing the Virtual Camera Target (DCT). Application software with Graphical User Interface is designed and implemented to allow users to select the required parameters of the imaging system and the required features of the resultant integral images. The software tool that is based on the developed techniques and employing OpenGL is useful to simulate the imaging systems, tune their parameters before the actual implementation of these systems, and as a result, save time and materials when designing these systems.

The introduced techniques and the software tools are faster, more effective, and cheaper original methods to help in optimising both the integral imaging systems and the quality of integral images. These software tools based on the new techniques can be used on a wide range of devices and platforms because these are employing the portable Application Interface OpenGL. With these methods, integral imaging systems are simulated, and optimised; good quality static and animated integral images were created.

Acknowledgments

I would like to express my thanks to Professor Amar Aggoun for the great support he has given me during this research.

Thanks to Dr Emmanuel Tsekleves for his constant help and support.

Thanks to Dr Maysam Abbod for his wholehearted help and support.

Special thanks to Professor Luiz C. Wrobel for his help and financial support.

I thank the School of Engineering and Design at Brunel University for providing the funding for this research.

Many thanks to the employees in CEME Company, East London, and my colleagues in the High Speed Sustainable Manufacturing Institute who helped me and participated in the assessment test and the integral images evaluation process I conducted in CEME Company.

My thanks to my wife Nesrin Eshtay for the encouragement she has given me.

Contents

List of Figures

List of Tables

List of acronyms

2-D	Two- Dimension
3-D	Three-Dimension
AII:	Animated Integral Images
AIST:	The National Institute of Advanced Industrial Science Technology
AOV*:*	angle of view
API:	Multi-Platform Application Interface
APS:	Application Software (DCTarget)
c:	camera location (DCTarget)
CCA:	Camera Central Axis
CCD	Charge Coupled Device
CGH:	Computer-Generated Holographic
CGIP:	Computer-Generated Integral Photography method
COMPSAII:	a 3-D computational Synthetic Aperture Integral Imaging
CP:	Centre Point
CT:	Camera Target
curv:	the value that measures the radius of the lenslet curvature
DCAJ:	Digital Content Association of Japan
DCTarget	Displacing the Virtual Camera Target
DIVGL	Dividing Image Volume using OpenGL
f :	focal length
f:	forward vector (DCTarget)
F:	the real distance between the image plane and the pinhole
f32:	a special type of real variables defined in Irrlicht
FII:	Final Integral Image
Fl:	focal length of the lens (DCTarget)
FPD:	Flat Panel Display
FOV:	field of view
fn:	file name
fps:	frames per second

GUI:	Graphical User Interface (DCTarget)
h:	the number of vertical pixels in the display screen (DCTarget)
hl:	height of the image plane or the screen (DCTarget)
HDTV:	High Definition Television
HMI	Human machine Interface
IIs	Integral Images
IP:	Integral Photography
ISTM:	Interpolative Shading Technique Method
LCD:	Liquid-Crystal Display
LCLV:	liquid-crystal light valve
Left plane:	the left vertical clipping plane
LPS:	Light Point Sources
M:	the maximum number of choices the viewer can chose
m:	indicates the positions of the image plane (DCTarget)
m:	the required projections for each object point in the scene (DIVGL)
MaxW:	the maximum weight of the choices
MD2:	MD2 files format
MI:	Multi-view Imaging
mode:	specifies the mode of the projection
model_mode:	specifies the model mode
MOS:	Mean Opinion Score
MR:	the resolution of the final image
N:	the number of horizontal pixels in the elemental image (DIVGL)
n:	the number of horizontal pixels under a lens with the width of Pt (DCTarget)
N:	the total number of effective lenses (DCTarget)
N:	the total number of viewers
n:	the total number of lenslets (DIVGL)
n2:	the refractive index of the lenslet
Na:	The numerical average
NHK:	The Science and Technical Research Laboratories in Japan
ob2:	Object 2
ob2'*:*	Object 2 virtual image
OpenGL*:*	Open Graphics Library
P:	the real width of the lenticular lenslet or the diameter of the micro lens

p:	the width of the pixel (DIVGL)
p:	Width of pixel (DCTarget)
p0:	point from the object ob2 in the scene
p0':	point from the object ob2 image ob2'
PAP:	a pinhole array on a polarizer of liquid crystal display LCD
PC	personal computer
PID:	Processing of Images for 3-D display
pitch:	the value that measures the pitch or the width of the lenslet
pos:	the vector components of a point from the object
Pov-Ray:	Persistence of Vision Raytracer.
PP:	the coordinates of a projection point on the aperture
Pt_l:	Pitch of lens (DCTarget)
Q :	the image quality
QoE:	Quality of Experience
R:	the resolution of the sub-images (DCTarget)
R:	the original target point *t0 (t0x, t0y, t0z)*
r_x, *r_y*, *r_z*:	the projections of the distance between the initial camera target and the camera centre on *x*, *y*, and *z* axis
RGB pixel	red green blue pixel
Right plane:	the right vertical clipping plane
s:	side vector (DCTarget)
SAII:	the synthetic aperture technique
sc:	the resolution measured as a number of pixels per mm
shift:	the horizontal and the vertical shift the user apply to adjust the integral image on the display
slot_index:	the x-coordinate of the frustum centreline
SLM:	Spatial Light Modulator
SPH:	used for spherical lenses
STRL:	Science & Technology Research Laboratories (Japan)
t_zerox, t_zeroy, and t_zeroz:	real variables to hold the x, y, and z coordinates of the initial position of the camera target
thick:	the value that measures the thickness of the lenslet
top:	the coordinate of the top horizontal clipping plane
up:	upper vector (DCTarget)

UNS:	Ubiquitous Network Society
Vf:	the final coordinates of the intersection point between the image plane and the ray that started from the projection point and passed through the object point and the lens array *x, y*
VNi:	the number of viewers who selected the choice *i*
w, b, y, r, and *g*:	shown the rays with the colours brown, blue, yellow, red, and green
w:	the number of the horizontal pixels (DCTarget)
Wab:	the width of the clipping window of the frustum
wl:	Width of the image plane or the screen (DCTarget)
Wp:	the weight percentage
Wi:	the weight of the choice
Yl:	the numerical value of the depth of the reconstructed point *ob2'*
Za:	the proportion of plane A' selected location that varies between *0* and *1*
Zab:	the absolute value of the distance between the projection reference point and the near clipping plane
zFar:	the point distance to the far-depth clipping plane
zNear:	the point distance to the near-depth clipping plane
α:	horizontal rotation angle (DCTarget)
α:	rotation angle
β:	vertical rotation angle (DCTarget)
θ_0:	the maximum value of the angle θ
μ:	the number of lenslets
zv:	the Z-coordinte of the lens array vertex, it is set to be 0

List of definitions

3-D Holoscopic imaging: the methodology of generating and displaying spatial images with a 3-D effect so that a real volume is added to the viewed images.

3-D Max: a computer program used to generate scenes and create images in the 3-D space.

Anaglyphs: is the name given to the stereoscopic 3D effect achieved by means of encoding each eye's image using filters of different (usually chromatically opposite) colors, typically red and cyan.

Animated films: made of a series of static integral images using the application software.

Aperture: a hole or an opening through which light travels

Application software: a set of one or more programs designed to carry out operations for a specific application.

Array of microlenses: an array of spherical microlenses situated regularly in rows and columns within a lenticular sheet.

Autostereoscopic imaging: Is any method of displaying stereoscopic images (adding binocular perception of 3D depth) without the use of special headgear or glasses on the part of the viewer.

Bi-linear interpolation: an extension of linear interpolation for interpolating functions of two variables on a regular 2D grid.

Binocular: binocular vision is the vision in which creatures having two eyes use them together.

Blender: another program to generate scenes and 3-D objects in the 3-D space.

Bottom plane: the coordinate of the bottom horizontal clipping plane

Capture stage: the imaging stage in which the scene is captured and the first image is taken

Central Processing Unit: the hardware within a computer that carries out the instructions of a computer program by performing the basic arithmetical, logical, control and input/output operations of the system.

Clipping window: The region against which an object is to be clipped

Convergence: is created by the difference between the angles of the left eye and the right eye viewing axes.

Cylindrical lens array: an array of cylindrical lenses, each lens has a curvature surface and a plane surface

Electro-floating display systems: The image floating is an antiquated 3D display technique, in which a large convex lens or a concave mirror is used to display the image of a real object to observer

Elemental image: is the micro image that is formed by a single micro lens array.

Frustum: the volume that contains everything can be visible on the screen after perspective projection.

Holography: a technique which enables three-dimensional images (holograms) to be made

Image plane:	the plane where the micro-images produced by the microlense array are captured and saved.
Integral image:	the image that is created with the integral imaging technique.
Integral photography:	displaying a 3D image without the use of special glasses on the part of the viewer.
Interpolative shading:	technique for surface shading in 3D computer graphics, provides a better approximation of the shading of a smooth surface.
Irrlicht:	an open source game engine written in C++.
Lenslet:	are small lenses, can be spherical or cylindrical
Lenticular sheet:	a set of adjusting cylindrical or spherical lenses.
Maya:	a similar program to 3-D Max.
Microlense:	a single lens of the microlense array.
Microlens array:	an array of spherical microlenses.
Model matrix:	convert from object space to world space.
Model-View matrix:	the model matrix multiplied by the view matrix.
Multi-view imaging:	multi-view imaging scenario where a number of cameras observe overlapping, translated sub-images of a larger scene.
Object point projection:	the trace of an object point on the image plane when the ray of that object point hit the image plane.
Object point:	is a 3-D point with 3-D coordinates. Each computer-generated 3-D object is formed of a number of object points forming the geometric structure of the object.
Open source ray-tracing package Pov-Ray:	a ray tracing program which generates images from a text-based scene description, and is available for a variety of computer platforms.
OpenGL:	a cross-language, multi-platform application programming interface (API) for rendering 2D and 3D vector graphics.
Orthogonal image:	the image of an object in which the near points in the real object appear as near points in the image, and the far points appear far points in the image.
Orthographic projections:	a means of representing a three-dimensional object in two dimensions, in this method, the projection lines are parallel to each other.
Parallax:	is the effect of viewing with one eye a view of the scene that is inherently shifted to the view seen by the other eye.
Perspective projection:	provide additional realism by making objects in the distance appear smaller than objects close by.
Pickup stage:	the capture stage. The first stage in imaging in which the scene is captured
Pinhole array:	an array of adjusting pinholes and located on the same distance to each other in regular rows and columns. Pinhole can be real or imaginary tool to capture the image. A micro image is formed from each pinhole so that a cluster of micro images are formed to be displayed as a 3-D image in the replay stage.
Pixel value:	is the digital data that describe the Red, Green and Blue (RGB) colours of the pixel in the display or the capture array. Each colour is expressed with a byte of 8 bit.

Plano-convex lenslet: Plano Convex Lenses are Optical Lenses with a positive focal length, one surface is plane and the other is convex.
Plano-Convex cylindrical lenses: an array of Plano-convex lenslets.

Polarisation: A polarized 3D system uses polarization glasses to create the illusion of three-dimensional images by restricting the light that reaches each eye, an example of stereoscopy.
Pseudoscopic image: is the image that is created in the first stage of integral imaging by a lens array in which the object point coordinates are reversed in z direction, x and y coordinates are rotated 180 degrees around the centre of the microlens.
Ray-tracing technique: a technique for generating an image by tracing the path of light through pixels in an image plane and simulating the effects of its encounters with virtual objects.
Real system: Real camera that captures real scenes and produces images of the scene
Replay space: the space where the recorded images are displayed and displayed to viewers
Reverse tracing: is the virtual tracing of the ray that would hit its pixel in the image plane.
Screen resolution: is the intensity of the pixels in the screen. It is measured by dpi or dot per inch. The number of pixels that occupy on inch on screen is the resolution.
Semi-cylindrical array: Plano-Convex cylindrical lenses
Spherical lens array: an array of spherical lenses in which one surface is plane and the one is spherical.
Stereoscopic imaging: is a technique for creating or enhancing the illusion of depth in an image by means of stereopsis for binocular vision.
Sub-image: The micro image that is formed of the rays of lights after leaving a single spherical or cylindrical microlense.
View matrix: the matrix that converts a point from world space to camera space by multiplying the point coordinates by that matrix.
VRML2: file format containing the object mesh and animation to a scene file with the integral imaging format.
XML: The configuration file can include the essential initial settings of the application such as the default model and texture, the welcome message etc…

Chapter 1
Introduction

1.1. 3-D Imaging

3-D imaging or 3-D Holoscopic imaging term used in this thesis means the methodology of generating and displaying spatial images with a 3-D effect so that a real volume is added to the viewed images. Therefore, the concept of 3-D imaging should be distinguished from the 3-D imaging that is used to represent images in the 3-D world on 2-D display devices. 3-D Holoscopic imaging methodology uses the principal of "Fly's eye" and hence allows natural viewing of objects (i.e. fatigue free viewing) [50] [103] [90] [83].

1.2. 3-D Imaging systems

Various types of imaging systems are developed or under development, these include two main groups: stereoscopic and autostereoscopic imaging systems. Stereoscopic imaging systems require glasses to view the 3-D effect of the images, whereas, autostereoscopic imaging systems do not require glasses and the Integral Images (IIs) can be seen with naked eyes. Examples of stereoscopic imaging systems are anaglyphs, polarisation, and time sequence. Examples of autostereoscopic imaging systems are holography, tracing systems, and imaging systems based on lenticular sheets. The main categories of the systems based on lenticular sheets are multi-view imaging and integral imaging.

The integral imaging principal was first suggested by Lippmann in 1908 [18] [67] [89] [84]. The idea of integral imaging is to capture the scene with an array of microlenses so that each microlense forms its own sub-image on a common image plane, then the cluster of sub-images is mounted by a microlense array with the same characteristics of the capture microlense to be viewed. An image with 3-D effect and continuous parallax can be viewed from different directions. Lippmann's principal was the subject of a number of studies and lots of research. Various embodiments based on Lippmann's principal were developed and implemented practically with different flavours.

In the proposed techniques that are described in Chapters 3 and 4, the targeted 3-D IIs are generated based on the pixel values that are forming the image. The pixel values and the coordinates of the pixels on the 2-D recorded images are calculated by reverse tracing the rays from the pixel to the object point in the scene that generates the ray. The value of the pixel is equal to the colour value of that object point. Reverse tracing the ray is the virtual tracing of the ray that would hit its pixel in the image plane. The location of the pixel on the image plane is defined by the coordinates of the point that is supposed to be hit by the ray. The coordinate's calculation is based on the characteristics of the optical devices used to capture the image and object points coordinates. The quality of the displayed images is dependent on the display screen resolution, the higher resolution of the screen the higher quality of the IIs. Therefore, when the images are displayed on normal screens such as a PC screen, the quality of images is limited due to the relatively low resolution of the usual display screens. The limited resolution means that the number of pixels is proportionally low to the point that the area of the screen that is employed to display an elemental image contains a low number of the pixel. Thus, the elemental image cannot accommodate enough information and pixel values for the part of the scene that is imaged and stored in the elemental image; as a result, the quality of the displayed image on a screen is relatively proportional to the resolution of the display screen.

The resolution of the display device is a vital factor to display good quality IIs, then, if a higher resolution display device was employed, a better quality can be achieved. Nowadays, the available printers can print images with much higher resolution than the available display screens. The number of dots that can be printed in the same area is higher than the number of pixels the same area can contain. When an elemental image is formed, the object points are projected on the image plane and each elemental image is occupying an area with a number of pixels equals to that of the intended display screen. The elemental images that are calculated in the application software are formed in a way that the size of each pixel is considered equal to the size of a single dot. The number of pixels per inch is considered equal to the number of dots, therefore, the number of pixels accommodated in the same area would be higher, and as a result, the resolution would be higher.

Real-time integral imaging is not available in the achieved application; however, the research can be extended and the application software can be developed to be able to produce real-time IIs. To develop a system for real-time applications, the system should be rebuilt to include a physical part accompanied with a software part forming an embedded system. The actual application software simulates the capturing stage of the image. The scene is a computer generated virtual scene, whereas, in real life, the integral imaging camera should capture the real scene, store the 3-D II data in a memory to be replayed either directly or indirectly and the data should be able to be transferred through communication channels and replayed remotely. The suggested system can include a pinhole array, spherical lens array, or cylindrical lens array to capture the scene. The image is captured with a Charge Coupled Device (CCD) and stored in a memory. An interface HMI can be employed to select the required parameters of the system. The image data can be processed using a microcontroller to correct the pseudoscopic image and create an II to be displayed by a display device with the selected parameters. Computer processing of the images to produce IIs replaces some of the physical equipment in the real integral camera such as these that are needed to correct the pseudoscopic image.

1.3. Aim and objectives

This study is focused on simulating flexible integral photography systems, generating good quality 3-D static and animated integral images using the simulated systems, producing a portable specialist software tool supplied with Graphical User Interface to achieve these targets, and proposing original techniques and algorithms to meet these purposes. In addition, it is aimed to optimise the integral images generation process to be more accurate, less expensive, more effective and faster. In order to compare the new developed techniques and the resultant integral images to previous techniques and the integral images produced with these techniques, investigating the previously developed techniques and methods in the field of computer-generated integral images, and implementing the closest technique are required.
The ideal scenario in generating 3-D images is the one that provides a perfect image by including information on the recorded image about the view of the scene from every angle. However, the best that can reasonably be achieved today is a less rigorous approach and information reduction is a key factor [8]. A trade-off between the size of information and the images accuracy can be applied in the case of generating 3-D images. Therefore, the techniques proposed in this thesis focused on reducing the size of data needed to be collected and processed to generate 3-D integral images (II), whereas, maintaining a high quality of the resulting 3-D IIs. The new developments on computer applications allows Lippmann's

principal of generating IIs to be implemented and simulated using the computer graphics applications. A number of studies were carried out to generate IIs using computer applications and methodologies based on Lippmann's principal. However, the existing methodologies are in need for more developments and enhancements to render better quality 3-D IIs with fast and effective computations. Using new and different technologies to produce 3-D IIs is useful to investigate the features and the benefits they can add to the process of producing such images. For practical implementation, it can be useful to produce application software to create 3-D IIs instantly starting form computer generated scenes so that the application is portable between different operating systems and able to deal with various image formats and different software programs. Computer generation of integral images is a simulation process of the real systems devoted to implement 3-D IIs; therefore, it can be useful to design special application software to simulate these systems so that the application helps the designer to select the parameters of the system devices and generate the required 3-D images based on the selected parameters. Therefore, this thesis is meant to focus mainly on the mentioned goals in an attempt to add some new features to the field of computer generation of IIs. The area of Computer Generation of Integral Images is the precise area of this study, and the main aim of the presented research is the production of computer generated integral images using algorithms, methods and tools more convenient for faster-to-produce, better quality and easier-to-tailor integral images, with lower computational, time and material costs. In order to achieve the required goals, a development strategy was proposed to bring the research from the theoretical ideas to the software implementation of the suggested ideas using computer tools and finally the actual production of the 3-D autostereoscopic integral images. The suggested methods that were meant to produce IIs and the software application tool that was designed to produce such images can be useful to simulate the process of producing IIs with a real integral camera. In addition, the designed software tool that is based on the suggested method is able to produce IIs starting form computer generated scenes and animations. The images and animated scenes can be designed using computer applications such as 3-D Max, Maya, and Blender, and then imported by the application software. In the application software, the suggested algorithms are applied to the imported images to be converted to autostereoscopic IIs with 3-D effect. The application software is supplied with a Human Machine Interface (HMI). HMI allows the user to select the features of the desired IIs and the virtual characteristics of the devices that are supposed to capture and display the IIs in the simulated real system. For example, the type and size of the lens array, the focal length and pitch of the lenses, the resolution and the number of pixels of the targeted display screen etc. These characteristics are chosen to be identical to the characteristics of the available physical devices that are supposed to be used in the display stage. If the application is used to simulate a real 3-D integral imaging camera, the ability to tune and select the parameters of the devices can be a helpful feature for the experimental design of that camera as this feature gives the user the capability to optimise the parameters and characteristics of the physical devices that would be used. The process of selecting the suitable parameters to obtain the best image quality for the different available devices and the input images is a useful way to save time and materials in the attempts to design a real camera. The optimised characteristics of the devices can be decided with the help of the application software before the actual implementation or usage of these devices; also, the selection of these parameters is based on the availability of the devices. For example, the resolution of the display screen is chosen based on the resolution of the available personal computer (PC) screen with which the IIs are supposed to be displayed. The images data can be saved and then displayed on normal display screens such as the screen of a PC. The display screen should be covered with a spherical or lenticular lens array with the same characteristics that were selected in the application software.

1.4. Contributions to knowledge

Up to my knowledge, contributions to knowledge are as follows:

- Dividing Image Volume using OpenGL (*DIVGL*) algorithm for generating good quality computer-generated 3-D autostereoscopic integral images.
- Displacing the Camera Target (*DCTarget*) algorithm for generating good quality computer-generated 3-D autostereoscopic integral images with a faster and less complicated process.
- Flexible and user friendly application software to implement the above mentioned algorithms and produce the required integral images.
- A method to evaluate the quality of the integral images based on the subjective evaluation of each one of the factors that are affecting the image quality of an integral image and related to the method of generating the images.

The above contributions are explained briefly in the flowing paragraphs.

1.4.1. Dividing Image Volume using OpenGL (DIVGL) algorithm

DIVGL is a simple and fast method to generate IIs using OpenGL library, in this method, the scene is a virtual computer generated scene. The capturing stage is simulated with the assumption of using a pinhole array, spherical lens array or cylindrical lens array to capture the image. The object is partially projected on each pinhole or lens to form an elemental image. The problem with displaying II is the limitation in display device resolution. In *DIVGL* method, the elemental images are formed by projecting the object partially on the image plane so that the elemental image is the perspective projection of a proportionally small part of the space that holds the scene and then the elemental images are processed to correct the pseudoscopic image. When the image is replayed, the different parts of the scene are replayed and their rays are intersected in the replay space forming the autostereoscopic images of the objects. The higher resolution of the display device, the larger part of the scene projection can be accommodated in the elemental image and then the quality is higher, ideally, if each elemental image holds the image of the whole scene, the quality is maximised.

1.4.2. Displacing the Camera Target (DCTarget) algorithm

DCTarget method is based on *DIVGL* method. *DCTarget* is a simulation with orthogonal projections of the perspective projection that is used in the method *DIVGL* that it is in its turn a simulation of the real imaging process. In *DCTarget* method, the II is composed of pixels selected from a number of orthographic images. Each orthographic image is the orthographic projection of the scene on an image plane rotated with a specific angle equals to the rotation angle of the virtual camera target. Specific pixels are selected from several orthographic images and mapped to a single image. The IIs and videos are generated with the application software from computer generated models and animations. The method provides fast generation of 3-D stereoscopic IIs with good full colour quality. In addition to the advantages of OpenGL rendering, *DCTarget* method reduces the time and computational efforts needed to project the scene. A reverse ray tracing approach was introduced and applied, in which the rays calculated starting from the image plane pixels that are supposed to receive the rays from the object points and ending with the object points. With this approach, several drawbacks are avoided. The tool that was created and employed to implement and test the algorithms is called the application software. One of their would-be practical applications is the printed integral images.

1.4.3. Application software

The cross language OpenGL (Open Graphics Library) was selected to be employed in the application software. Using the multi-platform application interface (API) OpenGL provides the software with valuable advantages and important features for graphics applications such as the ability to interact with a graphical processing unit to perform graphical functions faster than software applications running on normal Central Processing Unit. In addition to the ability to rendering images faster than other APIs, OpenGL is portable and able to run on different platforms. The application software is portable and able to import different computer generated images and animations produced by different software applications and formed in different formats. The imported images are converted to 3-D IIs with applying two different algorithms. The first is Computer Generation and Rendering of IIs using OpenGL (*DIVGL*). The second algorithm is Computer Generation and Rendering of IIs by Displacing the Virtual Camera Target (*DCTarget*).

In order to optimise the quality of the displayed IIs, a technique was suggested to print the IIs that are calculated by the application software using a printer on a transparent thin film. The film is illuminated with either an artificial light or natural day light on the back, and then the lens array sheet is placed on the top of the film. In this case, the resolution of the screen should be selected in the HMI to be equal to the resolution of the printer that is supposed to be used. With this option, we can control the resolution of the recorded image to be higher than the resolution of the image captured in real life 3-D imaging, and then, images with high resolution can be printed. This technique can be useful to display the produced static autostereoscopic 3-D IIs in advertisement panels, stable images or other applications where the animation of the 3-D IIs is not required.

1.4.4. A method to evaluate the integral images

Several factors can affect the quality of the computer generated 3-D autostereoscopic integral images. Some of these are related to the devices and tools used to generate and display the images. Other factors are related to the method of generating the integral images. This method is an attempt to evaluate the quality of experience and express it with a numeric value calculated from the results of a subjective evaluation of the factors that can affect the quality of experience and can be related to the method or the algorithm used for generating the integral images. These factors are determined as the following five factors: comfort measures the comfort the image cause to the viewer. Crosstalk measures the smoothness of switching between the integrated images when the viewer moves from one position to another in respect to the displayed integral images. Parallax is a one of the integral mages characteristics, this factor measures how clear and noticeable to view the parallax feature in the integral image. Depth factor measures the depth perception of the image in the viewer point of view, the depth of the viewed scene on both the two sides of the display screen. View angle is the angle through which the viewer can view the integral image when moving around the display screen; the higher view angle allows higher freedom to the viewer for moving without losing the integral image view.

1.5. The thesis structure

1.5.1 Chapter 1

An introduction with lists of structure items, objectives, and contributions.

1.5.2 Chapter 2

Includes a literature survey in which the latest approaches, methods and theories about the subject of computer generation of IIs are accessed. The survey emphasises on the fundamentals and state-of-the art of the area of computer generation of IIs. The main concepts of the different techniques approached for generating IIs are defined, analysed, evaluated and compared to the techniques that are proposed in this thesis. The survey includes a brief explanation and analysis of a number of the existing research papers, methods and techniques in the areas of 3-D imaging and IIs generation that are related to the specific field of this study (i.e. computer generation of integral images).

1.5.3 Chapter 3

Explains the method of Computer Generation and Rendering of IIs using OpenGL (*DIVGL*) and tries to prove theoretically and practically the validity of the approach. In the practical implementation, a virtual pinhole array was considered as the capturing tool that is used in the capture stage. A spherical lens array or a cylindrical lens array can replace the pinhole array to produce IIs that are more accurate. Cylindrical lens array was simulated by considering a group of adjusting pinholes so that the aperture that simulates each cylindrical lenslet is a vertical gap with a minimal width and a length equals to the length of the vertical lenslet. The theoretical concept was applied and implemented in the application software. Good quality IIs with 3-D effect were produced and shown. In addition, other aspects and results are declared in Chapter 3. In the theoretical and practical implementation, several obstacles were encountered and resolved. In addition, Chapter 3 includes a replication of the method proposed and implemented in De Montfort University (The UK) by Graham Milnthorpe. The IIs were generated using interpolative shading techniques, which is a slow method. In order to speed up the process of generating IIs and produce full coloured IIs with high quality, the method was modified and implemented using OpenGL application interface in C++ environment.

1.5.4 Chapter 4

Explains the method of Computer Generation and Rendering of IIs by Displacing the Virtual Camera Target (*DCTarget*). The rendering technique is based on the technique *DIVGL* that is proved in Chapter 3. The validity of the method is theoretically proved and practically implemented with the assumption of using a pinhole array as the image-capturing tool. IIs that are more accurate can be rendered if a virtual spherical or cylindrical lens array replaced the pinhole array, however, in this case, more complexity is added to the calculations, and rendering the images is slower. An additional enhancement was proposed aiming to produce IIs showing the objects situated both behind and in front of the display screen to make the images appearing more realistic. The proposed method was implemented with the application software and a variety of examples of resulting images is provided. Animated 2-D scenes generated in 3D Max or Blender are converted to 3-D animated IIs using the application software. Good quality IIs were rendered and other enhancements are proposed and added to the resulting 3-D images.

1.5.5 Chapter 5

Describes the plug-in tool or the application software that is used to render the 3-D IIs based on the proposed methods. Application software user selects the required parameters and characteristics needed for the required images, then operate the software that is supposed to import the computer-generated scenes or animations, apply the mentioned algorithms on the image data, and then convert the 2-D computer generated images to IIs with 3-D effect. To make the plug-in tool portable and able to use with different applications, several software issues were solved. In addition, Chapter 5 includes explanation about the software, the code and the employed HMI, guidance about how to install it and use it, the connections with other applications, and the obstacles that were faced and solved.

1.5.6 Chapter 6

In chapter 6, a subjective quality evaluation of the generated integral images was explained. The subjective assessment has taken place in a research institute within an industrial company. The 23 viewers who participated in the quality assessment are highly educated with good experience and knowledge in the field of image processing and 3-D technology. The assessment was carried out on the images generated with the two introduced methods (i.e. *DIVGL* and *DCTarget*) and a third method introduced in [8]. The compared methods are aiming to produce computer-generated 3-D autostereoscopic integral images. The evaluation and quality comparison were based on the introduced evaluation method in which five criteria of the image quality were evaluated individually and then the overall evaluation is calculated. The results of the subjective assessment are analysed and the advantages of the introduced methods of generating 3-D images were proved. The feedback provided by the participant was also analysed in this chapter and results are shown. An analytic assessment is carried out on the speed and complexity of the introduced methods against other close methods. The advantages of the introduced methods over the other methods in terms of complexity and speed were demonstrated.

1.5.7 Chapter 7

Summarizes the conclusions obtained from the research and declares the extent to which the aims of the research were met.

Chapter 2

A Literature Survey on the Computer Generation of Integral Images

2.1 Introduction

In this chapter, a literature survey is carried out on the theories and techniques approached to produce computer generated 3-D integral images. The work that is explained in this thesis is placed in the context of other works that have been done in the field. Some of the papers, theories and studies on which this study is based are explained. The differences between these techniques and the work of this thesis and the advantages of this work over the previous techniques are highlighted. This survey implies on an attempt to declare how this work is added to these techniques and enrich the field of computer generation of integral imaging. The papers and techniques that are reviewed in this chapter are selected among others because they are the closest to what is done in this thesis.

2.2 Autostereoscopic 3-D images

Images with 3-D effect are more realistic and closer to the real world images as they create an influence on the human eyes similar to that created by real objects. In general, 3-D images and videos are preferred over the traditional 2-D images. There is growing evidence that 3-D imaging techniques will have the potential to establish a future mass-market in the fields of entertainment (television, video game) and communications (desktop video conferencing) [1] [74] [93] [95]. Adding a third dimension to the traditional 2-D images and videos was the subject of many different approaches and inventions. Viewers need to wear special glasses to be able to see the 3-D effect of several types of 3-D images and videos. The need for such a device can form a practical obstacle in some cases when the viewer is not prepared to watch 3-D images [96] [75]. For example, if 3-D images are meant to be used for advertisement. In order to implement free viewing 3-D display, different methods were introduced aiming to produce autostereoscopic images.

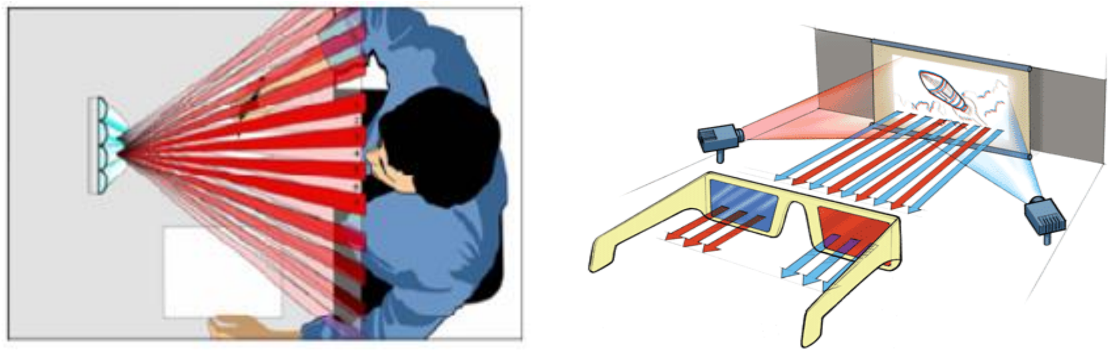


Figure 2.1: left: autostereoscopic image viewing, right: stereoscopic image viewing [1]

Stereoscopic technique is easy to realize, can produce large images, and high resolution. However, it needs glasses to evoke 3-D visual effect, provides observers with only horizontal parallax, only few number of viewpoints, and it causes visual fatigue because of the convergence-accommodation conflict. Autostereoscopic technique solves some of these problems. Several groups have demonstrated autostereoscopic 3-D displays [2] [99] [94]. True autostereoscopic 3-D display systems should have parallax in all directions and present images [1] [68]. Multiview Imaging and Integral Imaging (*II*) are popular methods for realizing autostereoscopic images. Figure 2.1 illustrates the difference between viewing an autostereoscopic image and a stereoscopic image. The former one does not need any glasses to be seen whereas, the later one needs a vision tool to be viewed (i.e. special glasses).

2.2.1 Multiview imaging technique

Multiview imaging technique is based on the idea of multiplexing at least two images of the scene taken from different points by either real cameras or virtual cameras. The resulting image is displayed under a sheet of microlens array or lenticular lens array. If the viewer is positioned in a correct location in respect to the display, each eye would see a different image at the same time, and then the viewer would be able to view the third dimension of the image. However, autostereoscopic display based on this principle does not produce a fully accurate 3-D representation of the scene [3] [98]. Multiview stereoscopic system employs lenticular lenses to play the role of a multiplexer with which the correct views of the scene are selected, displayed and specific views of the scene are seen by the left and right eyes dependent on the location of the viewer. In real life, a set of cameras is needed to produce such a collection of images and create a multiview stereoscopic system [20] [21] [22]. Figure 2.2 shows a lenticular lens array to view Multi-view images and the Multi-view image viewing.

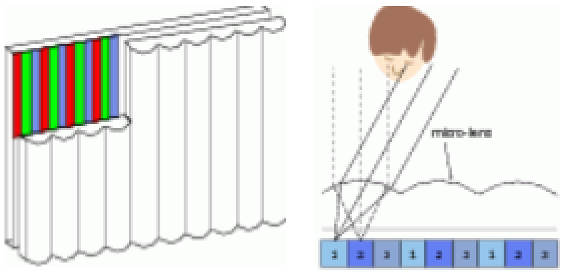


Figure 2.2: left: lenticular lens array, right: Multi-view image viewing [20]

2.2.2 Integral imaging technique

Integral imaging (II) is based on Integral Photography proposed by Lipmann in 1908. The methods proposed in II tended to improve the spatial resolution, enhance the limited depth of field, to expand the viewing area, or to solve the problem of the overlapping between the adjacent elemental images in the pickup stage. II systems are useful for other applications such as object recognition, the acquisition of the 3-D mapping of polarization distribution, and providing the input signal for electro-floating display systems [65] [97]. II can provide observers with true 3-D images with full parallax and continuous viewing points. II disadvantages include the limitation in viewing angle, depth of focus, and resolution of 3-D images. In addition, images produced by direct pickup II are pseudoscopic (depth-reversed) images.

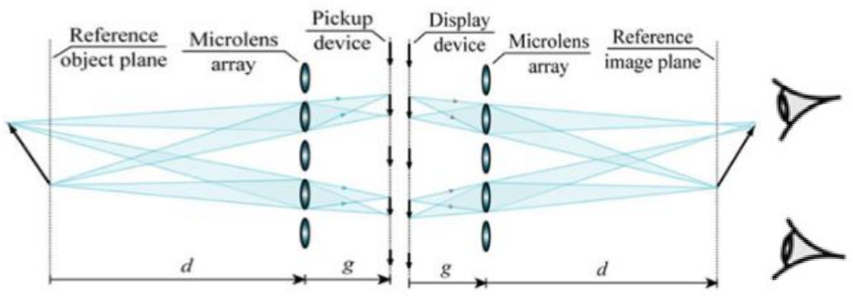


Figure 2.3: left: pickup stage of integral imaging process, right: replay stage [4]

Integral imaging technique is different to multiview technique as a single camera is used to capture a full volumetric optical model of the scene [17] [69]. In order to form an optical model of the scene, the microlens array or the lenticular lens array that is used in integral imaging production process samples the information coming from the scene in the capture stage and reconstructs the information in the replay stage. Integral imaging technique has many advantages over the multiview technique. The advantages include the ability to produce a higher quality of the 3-D images with the integral imaging technique. Figure 2.3 shows the pickup and the replay stages of a typical integral imaging process.

In the case of integral imaging technique, the viewer is able to view the scene clearly from any angle within the field of view that is related to the characteristics of the pickup and display devices. The viewer can move from one angle to another while the image smoothly changes. In contrary, multiview technique allows viewer to view the scene clearly from specific angles, and images jump when moving from one angle to another. In addition, integral imaging technique provides horizontal and vertical parallax.

2.3 Computer generation of autostereoscopic 3-D images

The mentioned techniques are supposed to be implemented using physical devices in the capture stage as well as in the display stage of the images. The computer generation of autostereoscopic 3-D images is the simulation of the real imaging that is implemented using computer tools and applications. Several types of real cameras have been designed to produce such images. Producing autostereoscopic images with these real devices is expensive and sometimes invisible. For example, if the 2-D scenes are computer-generated scenes and 3-D autostereoscopic images of these scenes are required, in this case, 3-D autostereoscopic images can be produced using a computer generation technique.

In this thesis, cheap and fast computer generation techniques can be employed to produce good quality autostereoscopic integral images are introduced and implemented. Other works in the field of computer generation of multiview and integral images are discussed and compared to the new methods.

2.3.1 Computer Generation of Multiview images

In the existing techniques, the distance between the viewer and the screen counts and does not allow the viewer to see a different region with each eye on a long distance due to the limited resolution of the screen. On a long distance, the angle of view of each eye is the same as the other eye, the eyes will be seeing the same pixels, and then the same view will be seen by each eye. This disadvantage is avoided in the techniques introduced in Chapters 3, 4, and 5 as the 3-D images are produced from the intersection of the rays emitted from the screen.

Referring to the method introduced in Chapter 4, the orthographic projections simulate the perspective projection of the scene with the same lenses array. The projection of the points on the image plane is equivalent to the perspective projection with the same microlens array.

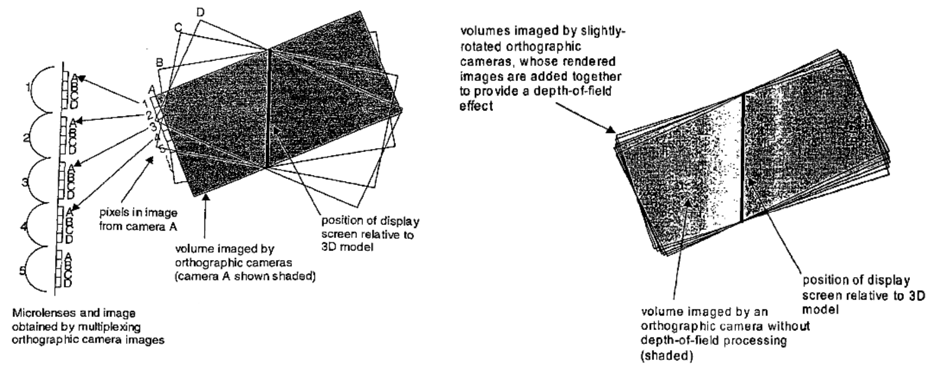
In the multi-view imaging technique, each eye views the scene from a different angle, therefore, the images can be combined incorrectly and that causes the image to be discrete. This disadvantage can be avoided when the images are real images formed from the intersection of the rays emitted from the pixels as it is the case in the introduced techniques, and therefore, the resulting integral images are more accurate.

2.3.2 Computer Generation of Integral imaging

G. Lippmann was the first to propose the idea of integral photography [18] [81] [87]. The method was developed by Ives [19]. 3-D Holoscopic imaging (also referred to as Integral Imaging) is a technique that is capable of creating and encoding a true volume spatial optical model of the object scene in the form of a planar intensity distribution by using unique optical components [4] [70]. II provides autostereoscopic intensity images with full parallax, free of any viewing device [5] [71]. In II technique, a microlens array is used to capture the image so that each lens views the scene from a different point and parallax information is recorded. 3-D image can be viewed when the parallax information is replayed. A number of methods and studies are focused on in the following paragraphs to highlight the differences to the approaches suggested in this study. The advantages and disadvantages of each method are stated. The reviewed papers and studies are selected based on the common points they share with the suggested methods.

2.3.2.1 Image processing for 3-D display

In 2008, Thomas and Stevens [3] patented a method to represent and display a 3-D model as a 3-D image called processing of images for 3-D display (PID). The images in PID method are several orthographic projections of the model at different viewing angles, multiplexed and presented under a spherical or cylindrical lens array so that different images are presented at different viewing angles. When the images are viewed from a specific viewing angle, the inventors claimed that the orthographic projections are demultiplexed. Parts from different orthographic images are viewed and the appearance of a perspective image is provided. Figure 2.4 shows PID integral imaging generating method.



A
B
C
D
1
2
3
4
5
pixels in image from camera A
volume imaged by orthographic cameras (camera A shown shaded)
position of display screen relative to 3D model
Microlenses and image obtained by multiplexing orthographic camera images
volumes imaged by slightly-rotated orthographic cameras, whose rendered images are added together to provide a depth-of-field effect
position of display screen relative to 3D model
volume imaged by an orthographic camera without depth-of-field processing (shaded)

Figure 2.4: PID integral imaging generating method [3].

With PID method, the need for several cameras is avoided; in addition, the quantity of computer processing needed to generate equivalent images and simulate the multiple cameras is reduced. However, due to the manner of capturing, multiplexing, displaying and viewing the images, PID method can be classified as a multiview imaging technique. Therefore, the multiview imaging disadvantages exist in PID method. These advantages include the flipping artefact that is caused by the limitation of the number of multiplexed images that is equal to the number of pixels located under each lens, and the lack of smooth switching form one view to another. In comparison to the method of Displacing Camera Target (*DCTarget*), *DCTarget* can be considered as an integral imaging technique in which the number of samples of the scene is much higher than that of the PID method as it is equal to the number of lenses. Hence, while the viewer is moving, there would be smooth switching between the views that are formed by the integration of the adjusting samples, and as a result jumping is invisible. The multiview imaging characteristics in PID method are expected to appear as the entire orthogonal image is multiplexed with the other images and their pixels are interlaced on the image plane. On the other hand, the width of the multiplexed image is higher than the width of the individual image and the resolution of the viewed images is lower than the resolution of the multiplexed image because the projections are minimised to a lower resolution. In the *DCTarget* method, only the selected pixels are recovered, picked up, and mapped to the Final Integral Image FII. In *DCTarget*, the width of each orthographic image is equal to the width of the FII; hence, the entire scene is integrated and viewed from each viewing point with a high resolution similar to that of the resulting image.

In claim number 14, the inventors claimed that the perspective qualities are formed from portions of the orthographic projections, the image is formed when the multiplexed images are demultiplexed. The perspective qualities in the *DCTarget* method are formed from groups of discrete pixels selected from the different orthographic images taken at specific angles corresponding to the angles of rays in a perspective projection. The image at the display is formed by the integration of the discrete projection points. With *DCTarget* approach, more accurate perspective images are formed as integrated images.

At a normal distance from the display screen, the eye of an observer receives rays from different images at the same time, and each eye can view a different combination of demultiplexed images. Stereo image exhibiting parallax and binocular vision occur when the two eyes view the scene from two different viewing angles. In order to provide such an image, each eye should view the scene from a different angle. In PID method, each eye sees a combination of orthographic images taken at different angles and the other eye sees the same or slightly different combination, which is not the ideal condition to produce stereo images with parallax and binocular vision. In the *DCTarget* method, stereo image providing parallax is verified by the virtual and real integration of the object points that allows the viewer to view the integrated objects from two different angles. In addition, the views seen in PID method are orthographic images, whereas, the views seen with the method of *DCTarget* are more conventional and natural because they are perspective images despite the fact that they are generated form orthographic images.

With PID invention, the rotation angles of the orthographic projections are meant to compensate the changing of the viewing angle. If the images taken were perspective, and the viewing angle changes, the viewer sees different perspective images, which is uncomfortable to the viewer. However, with PID method, the viewer sees different portions of several orthographic images for a specific viewing point, and then the changing viewing angle provides an automatic selection of which image is seen because the viewing angle is constant for each orthographic image but differs from one image to another. As a result, with this method, the viewer is supposed to see a perspective and natural image. The viewing angle is the angle between the direction of view and the screen.

In order to view correct and accurate images with PID method, the viewing angle of the viewer should be equal to the viewing angle of the orthographic image that should be seen from the specific viewing angle of the viewer. In other words, the accuracy and the quality of the display is dependent on the viewer's viewing angle that is dependent in its turn on the position of the viewer in respect to the display screen. In the *DCTarget* method, the quality and accuracy of the display is independent to the viewer position or viewing angle because the images seen by viewer are perspective-integrated images. If the way of selecting the image plane rotation-angles of the orthographic images, extracting the pixels, and mapping them to the FII are correct, the image perspective features are created. In such a way, the perspective projection of the scene is simulated, and the viewed images are perspective. In the *DCTarget* display, all the advantages of the integral images are available including the independency to the viewer location and viewing angle.

2.3.2.2 The simulation environment for generating integral images

3-D video systems have been for decades pursued as the video format of the future. Various approaches for providing a perceived depth have been invented [13]. Based on the method of *DCTarget*, animated images were produced using the application software. The animated images generated in separate applications such as 3-D Max are converted using the application software to animated images with depth perception. Each 2-D frame of the animation is converted apart to 3-D frame, and then the images are played in sequence to form an animated images or video display. Olsson and Xu [11] created simulation environment for a simple definition of complex scenes to form integral images and allow those integral images to be synthesized. The technique introduced in this paper will be compared to *DCTarget* technique to represent the advantage and disadvantage of each technique over the other. Figure 2.5 illustrates the block diagram of the mentioned simulation method, and an example of the generated integral image, the middle one is the 2-D image of the generated 3-D image (right).

A generic integral image description model was proposed to which different integral imaging techniques can be transformed and used by a simulation tool. The technique used in this simulation environment was ray-tracing technique. The simulation environment that allows using the model includes with two parts, an interactive tool with a graphical user interface

(GUI), and a rendering engine based on the open source ray-tracing package Pov-Ray. The interactive tool provides access to a generic scene description language to define scenes with different complexities, and allow user to modify the scene, generate the integral imaging sequence and store the sequence. In comparison to *DCTarget* method, the application software is used to implement the algorithm and produce the video stream of integral images. It is obvious that *DCTarget* is more flexible as a wider range of 2-D static and animated scenes can be converted immediately to images and video with 3-D perception, whereas, the mentioned system is limited to the scenes that can be created and modified by the user using Pov-Ray description language. In addition to the complexity of the system, generating integral images with ray-tracing technique is expensive in general, as it needs a large number of computation processes; in contrary to the *DCTarget* method that uses projection techniques, therefore, *DCTarget* technique is faster and cheaper. Moreover, ray-tracing advantages are provided in this technique despite that a projection technique is used.

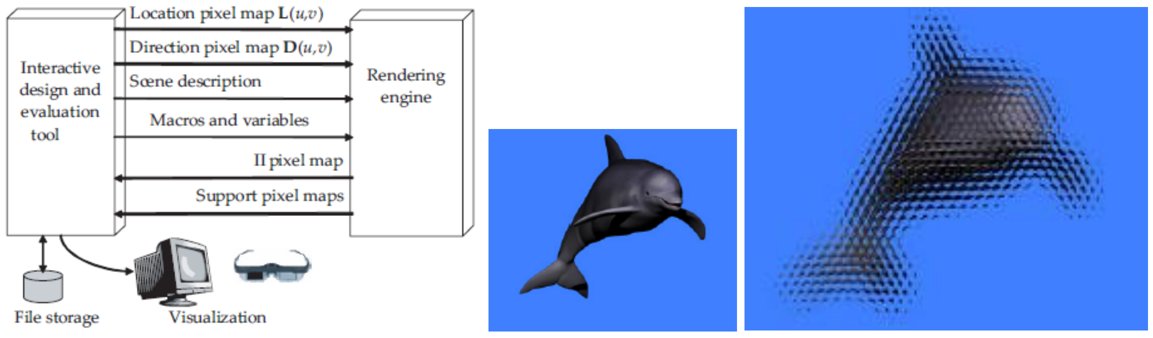


Figure 2.5: left: block diagram of simulation tool, right: generated integral image [11]

A simulation of the ray-tracing technique is implemented in *DCTarget*, in the pickup stage, the incident rays that are received by the pixels in the image plane are traced back to the object points. The value of pixel in the rotated image plane that holds the intersection point between an incident ray received by a pixel in the final image plane and the rotated image plane is the pixel value that is assigned to the pixel in the final image plane. Ray tracing from the pixels in the image to the object points is simulated using the orthographic projections of the scene object points. Another advantage of the *DCTarget* method is the application software that allows using more complex and accurate scenes with various types to be converted to 3-D integral images in a friendly environment. The GUI in *DCTarget* is more flexible and allows user to tune and change a variety of parameters in the system including the lens array parameters and the simulation mode.

2.3.2.3 Invert the pseudoscopic image to produce an orthoscopic image

The process of using a microlens array to capture an image of the scene, record it on a film and replay the scene using a microlens array causes the replayed image to be inverted in depth and pseudoscopic image is created. To produce an orthoscopic image, a second stage must be introduced to invert the pseudoscopic image and produce the required image. Davies and McCormick have developed an optical system to solve this problem [12] [88]. In addition, the system provided the possibility of imaging the scene and displaying a part of it in front and the other part behind the display screen at the same time. Figure 2.6 represents the system that is built by Davies and McCormick.

Sokolov proved that a pinhole array could be used to capture a scene and create integral images [15]. This model was implemented with a software model and integral images were produced and displayed on the flat surface of a microlens array [16] [77]. The pinhole array is simulated in *DCTarget* method as the tool used to produce integral images in a fast, cheap and efficient way with the ability to change the parameters of the pinhole array.

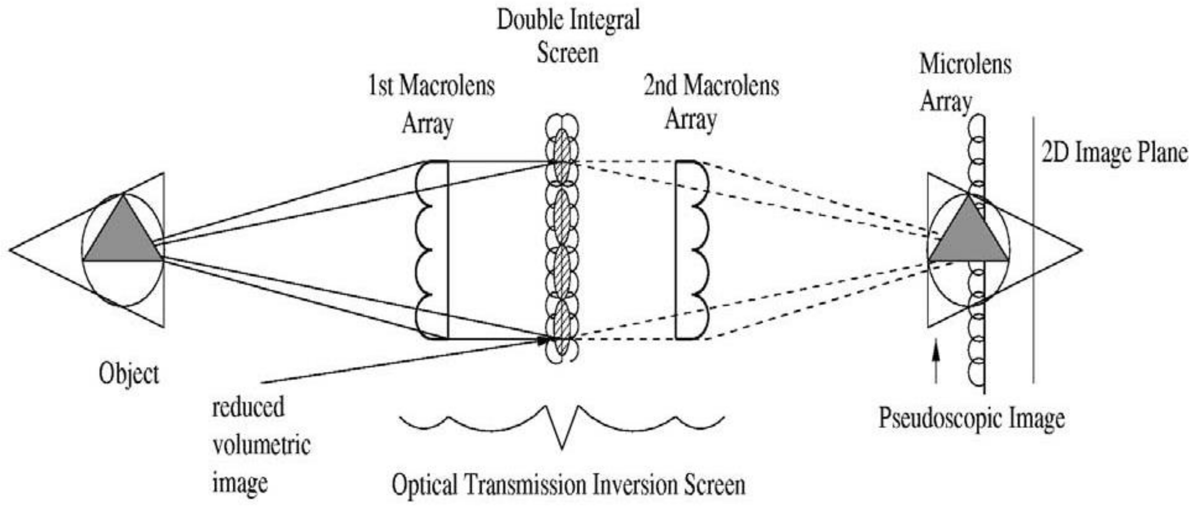


Figure 2.6: The system that is built by Davies and McCormick [12].

An integral imaging system was developed at De Montfort University by the Imaging Technologies Group, the system performs capture and replay in real-time capability [23] [24]. Milnthorpe designed a software model to simulate an optics system to render static and dynamic images in integral format [8] [14]. The system depicted in Figure 2.6 was employed in the previous studies to overcome the problem of pseudoscopic images produced in the first stage. The scene in this system can be replayed in front of and behind the decoding array. The model simulates the described system, each macrolens centre of the 2nd macrolens array is considered as a viewing point or a projection point from which the mode starts, and each macrolens centre behaves as a pinhole producing a pseudoscopic image of the original scene, whilst, the output of the 2nd macrolens is the aperture .

Starting from the projection points, each part of the scene is imaged in adjacent micro images to produce and encode intensity distribution that contains directional information needed to display integral images. A basic projection process is implemented on the vertices (i.e. object points) and the triangles forming the surfaces of the objects. Shading is implemented at a fundamental level as the basic calculations are carried out within the application software that is written for this purpose. Thus, the projected scenes are simple, the resulting integral images are less complicated and their quality is lower than those produced using more developed application interfaces such as OpenGL and DirectX in which other effects can be added to the scenes and therefore the quality of the resulting integral image is higher.

In *DCTarget* method, OpenGL is used and scenes created in applications such as 3-D Max can be converted to scenes with 3-D effect. Rays are traced form the aperture to the image plane to calculate the location in which the rays that are starting from the projection points intersect the image plane. Accurate calculations were applied in the ray tracing process taking into account the effect of the lenses on the ray direction and the intersection points in the image plane. The projection in this approach is applied to the pseudoscopic image in the second stage of imaging, whereas, in *DCTarget* method the projection is applied to the scene in one stage while the second stage of converting the pseudoscopic image to an orthoscopic image is implemented in the application software by mapping the pixels of the rendered image. For that reason, *DCTarget* method is easier. In this method the projection and ray tracing start form the projection point, whereas, in *DCTarget* the ray starting point is the pixel in which the value of the projection is stored and ended in the object points. The pixel values in *DCTarget* method are accurately calculated because the ray starts from the pixel; whereas the rays start from the projection points in the described method.

The number of projections can be less or greater than the projections required to project the scene because of the pixel dimension. A pixel in the image plane can receive more than one ray from different object points at the same time. Therefore, one pixel value can overwrite the others. That is due to the arbitrary distribution of the object vertices that can cause adjacent vertices to be projected to the same pixel when the angle between their rays is less than the angle that compass the pixel. These cases are avoided in the method of *DCTarget*, and as a result, *DCTarget* method produces a more accurate image with higher quality.

An object point in this method can be imaged in adjacent micro images if the location of the object point is suitable in respect to the projection points and the lens array. If the object point is located at a distance to the lens array closer than a specific distance, the object point will be imaged more than once in the same micro image. Whereas, an object point is not imaged in adjacent micro images if the object point is located at a distance greater than a specific distance, and therefore, flipping occurs and the quality of image is dependent on the location of the object points. In *DCTarget*, the object points are imaged for granted in adjacent micro images regardless their locations, and the flipping caused by imaging the object points in non-adjacent micro images is avoided, and as a result, the image quality is higher. *DCTarget* method in comparison with this method is faster, less expensive; the quality of the resulting image is higher.

2.3.2.4 Integral photography systems using a High Definition Television camera

The Science and Technical Research Laboratories in Japan NHK have developed a real-time Integral Photography (IP) system using a High Definition Television (HDTV) camera and an optical fibre array [26] see Figure (2.2). The integral imaging camera produced by NHK uses HDTV to capture each image field directly with a lens that projects images to CCD. The resulting image is an orthoscopic integral image at the display end. Gradient-index lenses are used to overcome the problem of overlapping image fields as they are supplied with optical barriers between the microlenses [27]. Figure 2.7 depicts the NHK real-time IP system.

In the paper published by Naemura et al [25] from the University of Tokyo, the authors proposed and implemented a method of synthesizing arbitrary views from IP images captured by the HDTV camera. In this method, a graphics system utilizing IP images from HDTV camera as an input was implemented and images from different perspectives were synthesised and rendered. Sixteen CCD cameras are used to capture light rays and the resulting 16 video signals are combined together to form the input of a graphics system. The light rays that pass a plane are stored as data of 4 dimensions. When the light ray data is acquired and selectively picked up from the data space, the image is synthesised as it is seen from an arbitrary perspective.

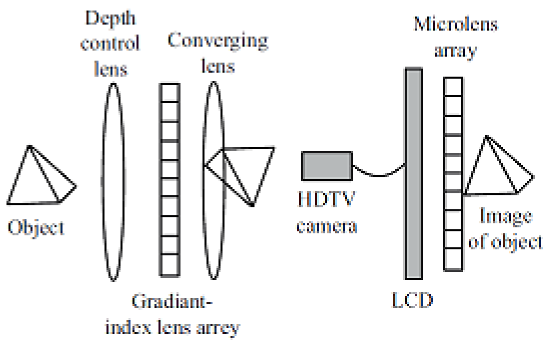


Figure 2.7, Diagram of NHK real-time IP system [26].

The system employs a microlens array to capture and display 3-D scenes. The rays for each direction that are passing through the centre of each lens with specific location form an elemental image. The elemental images are recorded and reproduced by selecting one pixel from each corresponding elemental image to synthesise the targeted integral image.

The method is complicated and relying on a large number of the images produced by the HDTV camera, the position of the viewer is critical, and the selected pixels are based on the position of the viewer. In *DCTarget* method, the integral images produced are based on a computer-generated model, the resulting integral image is fully produced with computer applications, and the quality of the image is independent of the position of the viewer. This technique was proposed, implemented and revealed in the paper written by Adelson, and Wang [30].

2.3.2.5 Displaying 3-D integral images

Various methods were proposed to display 3-D autostereoscopic integral images using computer generated elemental images. Seoul National University introduced the method of reflection type integral imaging scheme for displaying 3-D Images that is equivalent to the conventional transmission-type integral imaging method [29] [82]. In the later method, a convex lens array is used and the image is integrated by rays that are passing through the lens array, while a concave mirror array is used in the former method instead of the lens array and the integral image is formed by the intersection of rays reflected on the concave mirror array. Rays are emitted from an elemental image array that is placed in front of the mirror array or behind the lens array. Elemental images are generated by computer graphics tools. At the display stage, the elemental images are integrated at an object point. In order to display an entire object, the elemental image for each object point is generated.

In comparison with *DCTarget* method, the method of generating elemental images is expensive as the time needed to calculate the location of each pixel apart in the elemental image for the object points is relatively high. In addition to the complexities of ray tracing method, reflection type integral imaging scheme implies on other complexities due to the need for a beam splitter. Beam splitter is used in this scheme to help capturing the integral images without the obstacle of the display panel. To eliminate this restriction, the mirror array can be placed to be oblique rather than parallel to the display, this would increase the complexity of the calculations needed to produce the elemental images, and therefore, if such a position of the mirror array is required, the elemental images produced by computer graphics tools should be modified.

Reflection type method provides a limited view angle, while the view angle can be increased and maintained by tuning some of the parameters in the alternative transmission type Integral Photography with lens array. The calculations of the elemental images are affected by the position of the beam splitter, the size of the elemental image array, and other factors. If the size of the elemental image array is too large, restrictions on the display will occur due to the difficulties accompanied with placing the beam splitter, the mirror array and the elemental image array. The mentioned letter suggested a modified reflection type integral imaging approach to go around the problem of the large size of the elemental image array in the system. In the suggested approach, the concave mirror array is placed on a concave surface.

In this approach, the elemental image array would be more intensive and higher resolution would be required to locate the elemental images on a smaller surface than the one required for the flat mirror array, or the size of the mirror array should be larger to keep the same resolution and the size of the elemental image array. In addition, the calculations of the elemental images would be modified to fit in the new system, and additional calculations should be added to map the pixels of the elemental images to the new flat surface. In order to keep the object integration points in the correct positions, using concave mirror array implies on the fact that the original elemental images would be placed on different x, y and z coordinates rather than on the same surface when the flat array is used. Therefore, additional calculations are needed to map them to the same flat surface so that the implementation of the method becomes visible. As a result, both the two suggested reflection type and the modified reflection type of the integral imaging are very expensive, complicated and able to be replaced with the transmission-type physical implementation in which lens array is used.

In Japan, governmental and independent bodies recommended and proposed R&D related to the 3-D technology to be conducted as one of the strategic programmes to achieve the Ubiquitous Network Society (UNS), and Universal Communication. NTT DoCoMo developed an LCD-3-D display using a slanted lenticular lens producing high-density directional images. Toshiba produced a flatbed 3-D display showing 3-D images. Integral imaging, micro-lenses, and software to create the 3-D image were used. Sanyo employed a step barrier method with multiple parallax images. (STRL) of NHK developed 3-D display with LCD using an electro-holography method using high order refraction to overcome two problems: aliasing and narrow viewing zone. They were trying (2006) to develop 3-D video system using integral photography method. The National Institute of Advanced Industrial Science Technology (AIST), Keio. University and Burton Corporation have developed a "real 3-D" display in space, while most of the 3-D displays were pseudo-3-D images based on 2-D planes using the human binocular disparity. 3-D consortium released the 3-D contents and safety guideline called "IWA3" and conducted a survey about the growth of the 3-D market showing the market in Japan will grow rapidly. Digital Content Association of Japan (DCAJ) has conducted a survey on 3-D contents. The survey report predicted a huge market of the 3-D display using Flat Panel Display (FPD) with broadband penetration by 2015 [62].

A system proposed by Choi and Okano [35] aimed to build display devices to convert between 3-D and 2-D. The proposed method here solves the problems of a previous approach that needs a huge lens and a large space by adopting (PAP) a pinhole array on a polarizer of liquid crystal display LCD. This method enhances the optical efficiency in the 2-D display mode ten times more than the use of pinholes on a mask. In this method, Light Point Sources LPS are generated from a parallel light and lens array, the liquid crystal switches between 3-D and 2-D modes by eliminating and generating the LPS. The LPS can be formed by the light passed through the small apertures while the light is blocked elsewhere. Thus, there is no need for the large size and therefore, the size would be reduced.

The optical efficiency would be low as only a small amount of light can transmit through the small apertures. On the other hand, the pinholes should be removed when in the 2-D mode in order to display high quality images, while the pinhole panel cannot be removed electrically for higher optical efficiency. To solve the problem, this system uses PAP instead of the pinhole array, when the induced light is polarized orthogonally with the PAP, the PAP is a pinhole array and the 3-D mode is applied.

2.3.2.6 The field of view of integral imaging systems

The field of view (FOV) of integral imaging systems at the capture stage is limited by the aperture of the system; however, several studies were carried out in attempts to increase FOV and viewing angle. Stern and Javidi from University of Connecticut proposed a 3-D computational Synthetic Aperture Integral Imaging (COMPSAII) technique to increase the FOV and the viewing angle. The FOV angle determines the maximal viewing area or the range in which the viewer can move laterally [33]. In order to increase the aperture, the study suggested creating a relative motion between the object and the system, in other words, moving either the object or the system. In the capture stage, the object is scanned laterally from different locations and the resulting images are combined together with a computational generated method to form an integral image as if it is taken with a higher aperture system.

The FOV for an integral image is limited by the exit angle [34] [76]. The exit angle is selected so that the elemental images that are formed by each lenslet do not overlap and the exit angle is equal to the maximum angle the viewer can move in front of the display screen while viewing the image. The limitation of the exit angle value imposes limitations on the object size that can be entirely imaged without overlapping the elemental images. In addition, the FOV is limited by the finite aperture of the optical devices such as the pickup lenses that are employed to focus the images behind the lens array. Degree of limitation is dependent on the viewing point; the further viewing point from the lens array is the more limitation.

In SAII technique, the viewing point is supposed to be at infinity and on this basis, the computer reconstruction is obtained by sampling the II on a grid with lattice constant of the pitch size. The viewing angle is the field from which the whole object can be seen. With SAII technique, the study suggested that the viewing angle and the FOV are increased by increasing the aperture of the system that can be implemented synthetically by taking multiple exposures while moving the II system relative to the object in a perpendicular plane to the optical axis. The image formed by combining the II captured by scanning the object is equivalent to the image that would have been formed by a system with enlarged aperture. Moving the elements of imaging system implies on mechanical restrictions and difficulties and therefore, other limitations would be added to the imaging process such as the limitation on imaging a moving object or animated scenes, then the system should be very fast to image moving objects. In *DCTarget* method, the capture stage is computationally simulated, therefore, the target of using a system with an enlarged aperture can be reached by changing the characteristics of the lenses (i.e. decrease the thickness of the lenses and increase the number of the individual images or increase the lenslet lattice pitch). However, *DCTarget* method is meant to form integral images of scenes generated by 3-D computer graphics applications such as 3-D Max.

Another attempt to enhance the viewing angle of a 3-D display system was proposed and implemented in Seoul University. The system was a full parallax Computer-Generated Holographic (CGH) with an enhance viewing angle implemented by combining an integral lens array and colourized synthetic phase holograms displayed on a phase-type spatial light modulator. Holography is a 3-D technique that provides images with high resolution and sufficient depth information, but holography systems are complicated and the images has low viewing angle and requires a wide bandwidth, whereas II provides continuous viewing when the viewer moves, and full parallax without the need for special viewing devices. Computer generated hologram is made by a computer application that calculates and produces the image of an imaginary object. Spatial Light Modulator (SLM) is used to implement the amplitude or phase modulation of the CGH. The diffraction efficiency of the phase modulation is proportionally high which is an advantage added to having variety of applications [45] [72].

Real-time holographic display became possible due to the facilities offered by the recent technology development including the fast computers and advanced computational methods. The new technologies provided fast computation and holographic representation of 3-D data. On the other hand, some disadvantages are still accompanied with real-time holography such as the limited bandwidth of the communication and electronic devices. In *DCTarget* method, the computations needed to form the images are reduced by selecting one pixel for each micro image from a single orthogonal projection of the scene and repeat that for a limited number of projections to form the whole image. Alternatively, the scene must be projected a number of times equal to the number of micro lenses in the display device. In addition to the advantages II provides, producing II with *DCTarget* method is less complicated than the suggested holographic method, faster and less information is processed. The authors reduced the information content of the holographic patterns by introducing a Fourier-transformed synthetic phase hologram for an autostereoscopic 3-D image display system [46].

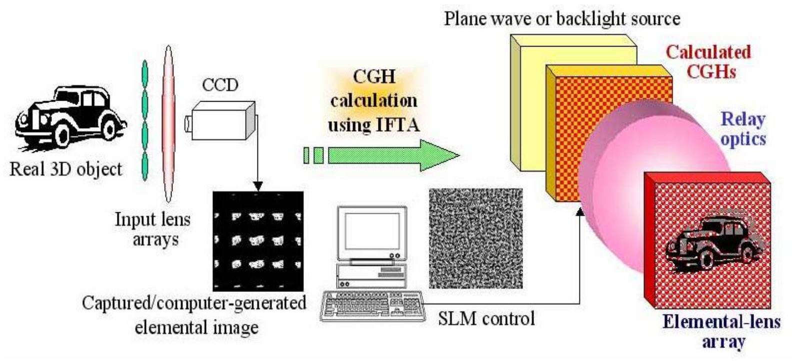

Plane wave or backlight source
Calculated CGHs
CCD
CGH calculation using IFTA
Relay optics
Real 3D object
Input lens arrays
Captured/computer-generated elemental image
SLM control
Elemental-lens array

Figure 2.8, Schematic diagram of CGH 3D display system. [46].

The researchers in Seoul University implemented a dynamic autostereoscopic 3-D display system using colour-dispersion-compensated method to produce full-colour 3-D images [47]. *DCTarget* method produces full-colour display integral images, the information content of the images including the colour information is derived from the computer-generated scenes and the colours of the images are kept with the same quality. However, the mentioned systems proposed by the authors have a small viewing angle, limited bandwidth and low-resolution problems. In the proposed CGH display system, a computer-generated 3-D image can be used as an input. The CGH is calculated and processed for full colour and then loaded to a spatial light modulator. Relay optics are used to carry out Fourier transformation and magnification of the phase modulated information that are outputted from the previous process.

Finally, a lens array is used to display the 3-D integrated image with a large viewing angle without the need for glasses to view the 3-D effect. In comparison with *DCTarget* method, the system and the devices needed are complicated and the computational costs are higher, whereas, the resulting images can have a similar or higher quality and viewing angle. In CGH system, the viewing angle is dependent on the resolution, the higher resolution the higher viewing angle, in CGH the pixel pitch is smaller than the normal LCD display device pixel pitch, therefore, the elemental images are higher resolved and the viewing angle is higher. In the integral imaging systems, the elemental images are calculated by ray tracing methods taking into consideration the pitch of the lens and the size of the pixels. On contrary, in CGH system, the elemental images are calculated by the scalar diffraction theory.

Beside the advantage of increasing the viewing angle, CGH system has disadvantages such as the complexity of calculations, the complexity of building the physical system, and the difficulty of the simultaneous real-time processing from the pickup stage to reconstruction and display stages. Using *DCTarget* method, the dimensions of each elemental image as well as the number of pixels in each elemental image can be calculated so that the elemental images are highly resolved. The calculated elemental images can be stored and processed to generate integral images with high resolution and as a result, a high viewing angle can be theoretically achieved. However, the limited resolution of display devices imposes limitations on the viewing angle of the displayed integral images generated by *DCTarget* method. The real-time calculations and processing of *DCTarget* images is simpler, and therefore, it is faster than the calculations used in CGH method. In addition, the implementation of the *DCTarget* system is easier because all the stages of the system apart from the display stage can be implemented using computer software applications.

The paper written by Jang and Javidi [33] suggested a technique in which the field of view is increased. In Integral Imaging technique, if the intensity and direction information about the rays were recorded in the device, the image would be reconstructed by regenerating the rays with the same direction and intensity information. If the directions of the reconstructed rays are the opposite of the recorder rays, a pseudoscopic real Image will be formed. If the directions of the reconstructed rays are the same as those of the recorded ones, an orthoscopic virtual image will be formed (e.g. the hologram).

In II, CCD device is recorded with either pinhole array or a lenslet array. Each pinhole or lenslet samples the ray information at its location. To reconstruct the image from the recorded elemental images requires a Spatial Light Modulator (SLM) such as LCD, and a pinhole array to select the proper ray directions. The pinhole samples a limited amount of the ray information that is determined by the lenslet pitch, therefore, the maximal viewing resolution of the reconstructed image is limited by the Nyquist sampling resolution. The pitch cannot be chosen arbitrary because of diffraction. The optimal lens size was suggested to be 1 or 2 mm in ordinary viewing circumstances. A no stationary lenslet array technique was suggested to overcome the Nyquist resolution limit determined by the spatial sampling rate of the lenslet array. The lenslet arrays for the pickup and display are synchronously vibrated in front of the CCD and LCD devices. CCD and LCD are foxed. The spatial sampling rate would be increased. Fast CCD and LCD devices are required to represent the no stationary elemental images. This technique also increases the viewing angle of the reconstructed 3-D image.

In the synthetic aperture technique (SAII), the lenslet arrays, CCD, and LCD are vibrated synchronously to enlarge the effective II system aperture. This approach increases the field of view, and improves the viewing resolution without additional image processing except the time-averaging effect in the observer’s eyes. In the experimental approach, LCLV (liquid-crystal light valve) was used; this optically addressed device provides a higher resolution than the commercial LCDs do. This technique is suitable for moving objects or when the system is on move. The system should move in the same direction of the object.

2.3.2.7 Animated Computer-Generated integral images

Researchers at Seoul University proposed a Computer-Generated Integral Photography method (CGIP) [48]. Picking up process was replaced with computer generation to generate the elemental images in CGIP method. CGIP system is formed from a computer system and display devices, therefore, it is similar to *DCTarget* system in terms of simplicity, and cost effectiveness. The animation in CGIP is presented by the time-varying elemental images, whereas, in *DCTarget* method, the animation is implemented by the projection of each frame of the animation generated in the specialist computer applications such as 3-D Max. The integral images that are formed from the different frames are displayed in sequence with an acceptable frequency so that the viewer is able to view the animation without flickering.

Objects captured with a number of normal cameras can be displayed in quasi 3-D if each camera captures a scene of the view with a different depth. The CGIP method is realized with software tools; therefore, it shares an advantage with the *DCTarget* method that is the flexibility of changing the parameters of the system such as the elemental image size and shape. This is useful to analyse the systems, optimise the parameters of the system and as a result optimise the quality and characteristics of the integral images. The resolution of the integrated images that are produced by the CGIP is inherently limited by the parameters of the system devices such as the lens thickness and pitch. The CGIP process uses computer-generated images to be the source of the integral images instead of the process of capturing real objects. The resulting pseudoscopic image is corrected to be orthoscopic image by the method of rotating the elemental images centrosymmetrically so that the revered depth is corrected when it is displayed on the display screen. Converting the pseudoscopic image to orthoscopic image using computer applications is the digital alternative of the complicated and expensive real conversion methods using lenses and physical devices. In *DCTarget* method, the conversion is more effective and more realistic as the resulting image can be either virtual- orthoscopic, real- orthoscopic image, or virtual- orthoscopic and real-orthoscopic image appearing in front and behind the display screen. The two methods, *DCTarget* and CGIP use software tools to achieve such a conversion but *DCTarget* method allows the user to select the mode of display and select a specific plane in the scene to display the part of scene behind it as a virtual orthoscopic image and the part in front of it as a real orthoscopic image. Figure 2.9 shows the schematic of the CGIP system.

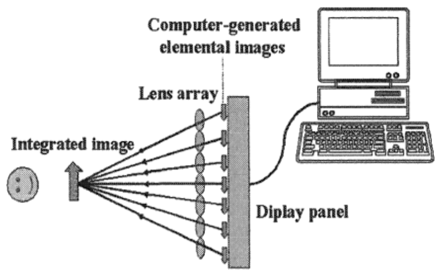


Figure 2.9, Schematics of the CGIP system. [48].

In CGIP method, the lens array used to capture the images of an object and create the 2-D elemental images of the integral mage is got rid of, and instead, the elemental images are generated by a computer. The x-y coordinates of the elemental images of an imaginary object are calculated and mapped to an array of 2-D information. This process is implemented on each point of the object taking into consideration the depth information that is more expensive and slower than the process that is carried out in *DCTarget* method in which a limited number of projections are applied and then the pixel values are selected from these projections in a faster and easier algorithm.

Another feature is added to the images created with CGIP method is the correction of the pseudoscopic image that results from the first stage of the image capturing. The correction is based on the algorithm of reversing the locations of the pixels that are forming an elemental image around the centre of that elemental image. However, at the display stage, the resulting orthoscopic image of this algorithm is viewed as if it is located in one side of the display screen. Whereas, in *DCTarget* method, an orthoscopic real image can be created with the possibility to display a part of the image as an orthoscopic real image in front of the display screen and the other part as an orthoscopic virtual image behind the screen. To implement such an image in *DCTarget*, the part in front of the screen of the pseudoscopic real image is converted to an orthoscopic real image, and then displayed on the same final integral image together with the part that is located behind the screen as an orthoscopic virtual image.

The image mapping process calculations in CGIP method start with the calculation of the centre of each elemental lens, then the virtual object is considered as a combination of planes with different depths so that the depths change along the depth direction. For each object point, a set of elemental image points are obtained and plotted. The object point coordinates x-y-z determine the centre of the corresponding elemental images, the number of the related elemental images and the points on the plotted image of the object point in question. The process is repeated for each object point until a set of elemental image is plotted. For a scene with more than one object, the process is applied to each points of every object; the hidden parts of the scene are considered and plotted despite that they should not be viewed; therefore, unwanted and unnecessary calculations are carried out. For this reason, CGIP method is not the best for multi objects scenes. *DCTarget* method provides the possibility to project the scene taking into consideration the hidden parts of the scene and plotting only the viewable parts of the scene from the angles within the viewing field of the viewer.

2.3.2.8 The software used to render integral images

Within the Prometheus project, Price and Thomas [43], a pinhole approach that will generate images in real-time is presently being developed and is the second software model developed by this research. The reduced content integral images produced, using a limited number of projection points, are sufficient for TV purposes.

The OpenGL software package was used in *DCTarget* method as the application interface to render the integral image content. POV-Ray software package that is based on the technique of ray tracing can be used to generate integral images with ray tracing algorithms. In Athens University, POV-Ray software was used to implement a method to generate integral images based on ray tracing technique [49]. The mentioned work at Athens University focused on the conversion of 3-D computer generated models to integral images. A technique was proposed to simulate the first stage of producing orthoscopic real and virtual integral images, which is the capture stage. The proposed technique provided the possibility of producing integral images for square and hexagonal microlens arrays. The quality and complexity of the integral images produced by this method are higher in comparison to those of the images produced by some other methods.

The first stage of integral images generation was simulated using POV-Ray ray tracing software package. In the capture stage, an imaging lens is employed to focus the image of the objects that is formed as a pseudoscopic image; the image in the image space can be captured by a lens array that creates the elemental images of the scene on the pickup image plane. If the lens array was placed in the image space between the imaging lens and the image of the object, the image that is formed on the pickup plane is virtual pseudoscopic, then the image is rotated with 180° to correct it and convert it to an orthoscopic image. The replayed integral image would be 3-D orthoscopic real image. If the object image that is formed with the imaging lens was located between the lens array and the imaging lens, the image information about this object that are recorded on the pickup plane represent a pseudoscopic real image. Then this image is rotated 180° around the centre of the elemental image and the image resulting in replay stage is a virtual orthoscopic image.

Figure 2.10 shows the single stage integral photography capturing for production of orthoscopic real images performed by the research group at Athens University. In *DCTarget* method, the virtual capturing stage is implemented with the assumption that a pinhole array is used to form the elemental images of the scene on the image plane instead of an imaging lens and spherical lens array. Therefore, the calculations needed to render the integral image are fewer, can be performed faster and easier which is a good advantage in the case of rendering real-time animated integral images.

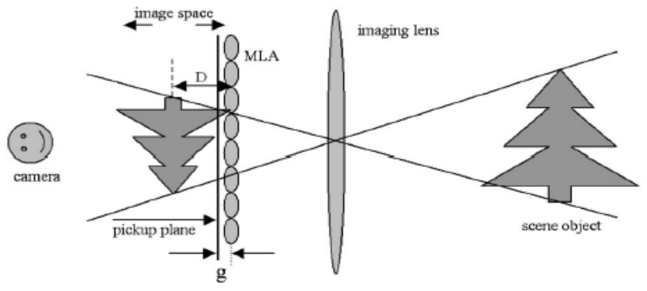


Figure 2.10: Single stage capturing setup for production of orthoscopic real images [49].

The technique used in *DCTarget* of orthogonal projection provides even faster and easier way to render the integral images. In addition, the software package used in *DCTarget* is OpenGL application interface and other related libraries instead of POV-Ray ray tracer package that is used in Athens University technique. OpenGL is an efficient application employed to render very good quality and complex scenes using mathematical techniques based on calculating the matrices that generate the scene. Therefore, calculating the integral images would be faster than the POV-Ray based technique of ray tracing in which each point is calculated apart, adding the need for more efforts and time to the integral images rendering process.

The quality and complexity of the integral images produced by *DCTarget* method are higher due to employing applications such as 3-D Max with OpenGL software tool to render the images. Using these applications instead of POV-Ray allows the designer to create more complex scenes and higher quality. In addition, with OpenGL the images are rendered faster which is a suitable feature to produce real-time animated integral images.

2.3.2.9 Lens arrays for 3-D imaging systems

The key components in the II technique described in reference [61] [59] are multiple arrays of lenses that relay, invert, and encode a range of views of the object. The microlens array was developed for a camera designed by Davies and McCormick [12]. This camera uses two-dimensional arrays of Plano-spherical lenses throughout. It is the first single stage large aperture camera to be used for directly recording, in transmission, images that replay with full parallax and orthoscopic perspective. A recording is made with a photographic plane or electronic image sensor placed in the focal plane of the lens array. To replay the 3-D image the light paths are reversed and the lens array generates light beams that intersect and form the integral image.

The image transfer screen is a pair of microlens arrays separated by the sum of their focal lengths. This inverts the depth and generates a pseudoscopic image. Real images of the 3-D object is formed by a large lenses array, 39mm pitch hexagonal lens and relayed to the encoding screen by a similar array. The recorded image is replayed when the light paths are reversed and a single-lens array is used as a replay screen to generate an orthoscopic image. To widen the field of view the aperture ratio is made as high as *f/2* (i.e. the focal length divided by the effective aperture diameter). Lenses can be formed by melting photo-resist, graded exposure in photo-resist, or formed as replica of an ion-etched master. The image transfer screen formed is a pair of large lens arrays formed by melting resist arrays on 9mm thickness glass substrates.

The encoding screen is the lens array placed in front of the photographic emulsion to record the 3-D image. Close-packed lens array is necessary to obtain maximum view ability in the reconstruction. Master array was made using ion-etching technique and replicas were made by casting resin on glass substrate. The replay screen is placed in front of the recording to reconstruct the 3-D image. When using the encoding screen to replay 3-D images, a variety of different size images can be displayed when it is required by selecting the appropriate pitch and size lens arrays; however, the lenses have large sag heights. Therefore, a diamond-machined master method was used.

2.3.2.10 Computer reconstruction of 3-D images

The computer-based image retrieval provides the ability to improve the quality of the image such as contrast, resolution, and brightness with numerical techniques. With his method, there is no need for special purpose devices to display the 3-D image such as high quality LCD or micro-optics [63] [73]. Okano proposed using 3-D integral images on LCD for 3-D television, and using gradient-index lenses to tackle problems such as orthoscopic - pseudoscopic conversion or interference between elemental images [13] [58]. The optical reconstruction method proposed in [34] and [74] may introduce a resolution limitation in the 3-D integral imaging. The optical integral imaging introduces different problems. These problems can be exceeded with using the digital 3-D image reconstruction method.

A computer-based integral imaging using high-density television camera was proposed in [25]. It is a simple method to produce a 3-D integral image. The system consists of a microlens array to form the elemental image array of the 3-D object, and a CCD camera to record the image array. The image is reconstructed using a computer by extracting pixels periodically from the image array. Image processing can be employed afterward to improve the image. The image data can be sent via the internet and displayed without the need for special devices. In the system, a camera lens is inserted in between the CCD camera and the microlens array to form the image array on the CCD camera. The magnification of the lens is adjusted so that the size elemental image array becomes the same as the size of the tip of the CCD camera. The computer reconstructs the image by extracting a pixel from the elemental image array every a number of pixels equals to the number of pixels existing in each elemental image. The resulting image is viewed from different locations using a microlens array. The resolution is determined by the resolution of the CCD camera and the number of the lenses in the microlens array. The image is later processed to improve the quality, contrast and brightness and reduce the speckle noise. A sequence of images can be reconstructed and animation can be created using an animation technique that allows the image information to be sent via the web.

The integral Imaging system gives a pseudoscopic image when operating in the standard conditions. The technique proposed in [65] [80] is a way to produce real, orthoscopic, undistorted, integral images by direct pickup. The technique is based on a smart mapping of pixels of an elemental images set. II technique works without incoherent light and provides auto-stereoscopic images without the need for glasses. The problem of overlapping between the adjacent elemental images in the pickup stage was tackled with the gradient-index microlens approach and the use of a barrier array [86] [91].

2.3.2.11 Depth of field improvement in the 3-D integral imaging systems

Refraction and other factors cause the limitation in the depth of field of 3-D integral imaging. Reducing the numerical aperture of the microlens improves the depth of field but that costs the decline in the spatial resolution. The amplitude modulation of the array of phase elements method can improve the depth of field with no deterioration of the spatial resolution [64] [79]. In order to improve the limitation of the depth of field of the lenses, in the past, methods based on the synthesis of the real and virtual image fields, non-uniform focal lengths, and aperture sizes were proposed.

The method proposed in [64] enlarges the depth of field without affecting the spatial resolution. For that method, the Equations that describes the diffraction behaviour of the pickup of the integral imaging system is extracted. The depth of field is determined by the so-called Rayleigh range. The size of pixel highly affects the depth of field and resolution. The authors of [64] stated that if the lenses were covered with a circular obscuration on the central parts with a diameter of sigma, the depth of field would be higher. The more the sigma coefficient for the cover the more the depth of field will be, but the smaller the light efficiency of the system and the more the optical aberration. Because of these reasons, they proposed to modulate the amplitude transmittance with the so-called binary modulator that provides 50% efficiency and doubles the depth of focus of the system. The binary modulation worsens the resolution of the points with low depth coordinate z of the object.

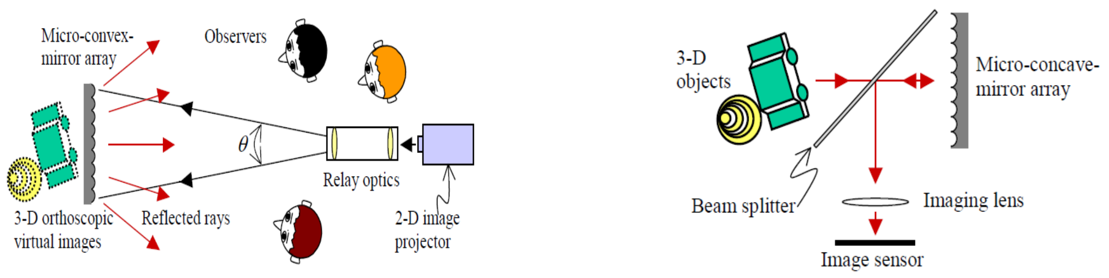


Figure 2.11: left, 3-D projection II using a micro-convex-mirror array, right, direct pickup of elemental images using a micro-concave-mirror array [66].

The authors stated that the technique proposed in [66] uses micro-convex-mirror array will solve several problems of the II including the limited number of pixels in the systems using microlens array, the limited viewing angle problem, and pseudoscopic to orthoscopic image conversion problem. The use of micro-convex-mirror array increases the viewing angle significantly because it is easy to make micro-convex-mirror array with a small *f*-number with negligible aberration. Orthoscopic virtual images are automatically displayed. Flipping-free observation of 3-D images occurs without the use of optical barriers because each elemental image is projected onto only its corresponding micro-concave mirror. Figure 2.11 shows an optical setup for experiment on 3-D projection II using a micro-convex-mirror array on left, and direct pickup of elemental images using a micro-concave-mirror array on right [66].

2.4 Summary

Several approaches in the field of computer-generated integral images were reviewed and compared to the techniques that are introduced in this thesis. The additional features in the introduced techniques that are added to the previous approaches were highlighted. The work in question was placed in the context of other works that are achieved in the same area of research. Other related studies were selected and explained briefly.

The 3-D autostereoscopic integral imaging field, and the computer generation of integral images approach were defined and the new study was placed in the context of the other works in the same field.

The selected studies were close to the suggested techniques that are implemented to produce computer-generated 3-D autostereoscopic integral images in one aspect or more. In each previous work, the technique was compared to the suggested methods in this study and the advantages and disadvantages of one over the other were highlighted.

It was proved that the new techniques have advantages over the previous work. In addition, the new methods are different to the previous ones and original. The need for the new techniques has been justified.

Chapter 3
Computer Generation and Rendering of Integral Images Using OpenGL (DIVGL)

3.1. Introduction

Integral Imaging (InI) is a method to acquire and display 3D images and movies. InI provides autostereoscopic intensity images with full parallax, free of any viewing device [4] [92]. InI is based on the integral photography (IP) technique that was proposed by Lipmann in 1908, in the capture stage the micro lenses array forms a cluster of elemental 2D images on an array of image sensor (Charge Coupled Device CCD).

A 3D image is reconstructed with the elemental images in the display stage using another micro lenses array. Each elemental image records information from a different perspective. The 3D image is reconstructed in the image space by the intersection of the rays produced by the micro lenses.

This chapter includes two main parts 3.2 and 3.3. In part 3.2., the model used is called Forward Projection rendering model. The method was used in the thesis titled “Computer Generation of Integral Images Using Interpolative Shading Techniques” [8] [78]. The method is implemented in this thesis with OpenGL in the environment of C++. The method was implemented in the mentioned thesis using simpler tools and less professional rendering application. The aim of the new implementation of the method is producing higher quality integral images to be compared with the integral images produced by the new methods that are introduced in this thesis.

In the part 3.3, the method called Computer Generation and Rendering of Integral Images with the Method of Dividing Image Volume Using OpenGL is introduced. Perspective projection mode using virtual cylindrical lens array, and perspective projection mode using virtual spherical lens array were implemented. Some computer generated integral images using the two types of virtual lenses are provided.

3.2. Computer generation of integral images with OpenGL using Forward Projection rendering model

The research with the title “Computer Generation of Integral Images Using Interpolative Shading Techniques” [8] was carried out in De Montfort University (Leicester) by Graham Milnthorpe. The fast computer generation of integral images is the main target of the mentioned thesis. This fast method is used for the generation of integral images in real-time. The ray tracing method is a slow process because the intensity of each pixel is calculated apart by tracing the path of light through it.

The alternative is the forward projection method whereby most of the pixel intensities are estimated by bi-linear interpolation. Thus, the time and the computation costs needed to calculate the pixel intensities in this method are obviously lower. Forward geometric projection technique is used to both capture the image and replay the pixelated surface.

In the capture stage, the image plane is a set of pixels employed to receive the projected object image and record it. A lens array is positioned on the image plane, and selected projection points are considered to project the object image on the image plane through the lens array. The object is located in the space between the lens array and the projection points. In the replay stage, a lens array similar to the one that is used in the capture stage is positioned on the film on which the image in the capture stage was recorded. The viewer can view the 3-D image through the lens array.

The viewer should be able to move and watch the 3-D integral image from different points without flipping. The integral image can be produced by using various projection points forming different perspectives. Two methods are used, and the number of the projection points is dependent on the selected method. In the first method, the projection points were calculated based on the depth of scene so that two projection points encompass the space imaged by a single lenslet. In the second method, the number of the projection points is selected to be equal to the number of horizontal pixels located under each lenslet.

The thesis tried to proof that the computer generated integral images can be produced using forward geometric projection. The technique of Forward Geometric Projection has few advantages over the traditional Ray Tracing technique. The rays in the Ray Tracing technique are extrapolated from the pixels through the pinholes and intersected with the object points. Therefore, some of the object points are imaged in the lenslets but not all of them. Whereas, in the projection technique, if the scene was represented with triangular mesh data, the pixel hits are calculated for the triangle corners projected on the image plane and then the pixel hits for the whole triangle points are calculated.

Another advantage is that the processing time is shorter, because in the ray tracing technique, each pixel hit must be calculated by tracing the ray and its intersection with the object is calculated, whereas, in the projection technique, the pixel hits are calculated with applying an algorithm. Two derived methods for producing integral image animation using interpolative shading were described. The first method is based on the forward projection through pinhole mesh model (pinhole model). The second method is based on the forward projection finite-sized aperture mesh model (finite-sized model).

In the pinhole model, an integral image is generated using a pinhole technique in which an object is created with all the triangles forming the object mesh. The points between the triangles corners are calculated to form the triangle’s sides and connect the vertices of the object to each other with the resulting lines. Then the algorithm that is used to generate the projection on the image plane is applied to each point of the object points including the generated and calculated ones. For each point of the object, the pinholes seen by the point are calculated. The pinholes are the vertex of the lenslets through which a ray from the object point is passed to hit a certain pixel. In order to calculate the pinholes covered by a point, rays from the extreme borders of the aperture are passed through the object point in question and then through a pinhole. The pinholes located in between the extreme passed pinholes are counted and considered for each project point, these are the pinholes seen by the point. For each point and each pinhole seen by this point, a virtual ray is passed by the virtual lens array through the pinhole, the refraction effect of the lenslet on the ray is taken into account, and then the pixel hit by the ray is calculated and its intensity is worked out. When all the pixels and their intensities are calculated for all the object points and all the pinholes, the total image is written and ready to replay.

In the finite-sized model, the same process of the pinhole model is repeated with few differences. In the pinhole model, only rays that are passing through the vertex of the lenslets are considered, and the location and intensity of the pixels hit by these rays are calculated. In the finite-sized model, the rays are extrapolated from each object point to a number of locations on the curved surface of any lenslet within the coverage of each point. The refraction effect on the rays and the deviation of the location hit by the rays on the image plane, the location of the affected pixels, and intensity of the affected pixels are calculated for all the projection points, all the object points, all the lenslets, and all the considered locations on the lenslets curved surfaces.

The scene is created with all its objects. The vertices of each object are found and connected to each other with lines. For each projection point, the lenslets seen by this point are taken into account one by one and the curved surface of the lenslet is divided to equally spaced lens points. Then the rays are transmitted through each lens point with calculating the refraction. The process is repeated for each lens point on each lenslet seen by the object point. At the end of all these calculations, the image will be written as a set of pixels intensities saved on the film located in image plane. The problems associated with these calculations were solved including the problem of integer and non-integer number of pixels under each lenslet. However, the image generated from this model has spherical aberration and defocus effects as for image captured using real optics. The pinhole model is faster because of the extra computation needed to generate the volumetric image in the finite-sized model.

Two approaches were used to carry out interpolative shading on the integral images. The first approach is implemented in the image plane on the points projected form the object space after linking them with lines to form the perimeters of the triangles forming the mesh of the object. The lines are calculated by allocating a number of points between each two corners of the triangle, and then projecting these points separately to the image plane.

The projected points on the image plane have no identities to state they are belonging to specific triangle or line. The interpolative shading is applied to these points in the image plane regardless the relation between each other or between them and the triangles they were belonging to in the object triangular mesh. The other approach requires drawing and saving of the perimeter points in object space before translating the whole perimeter. The projected points must be given an identification number so that the translated object point on the image plane can be identified by these numbers so that the perimeter and triangle to which it is belonging can be identified.

3.2.1. The Forward Projection Finite-Sized Aperture Rendering Model

In this model, the second approach is applied. The rays are intersected with the lenslet curved-surfaces and then refracted before hitting the pixels on the image plane. This model acts as a software model for a real finite-sized aperture integral imaging camera; the program includes variables to be changed to provide its design parameters. In order to produce the required integral images, an object file format VRML2 was developed and designed to contain all the required variables as input such as the optical variables, giving the user full control for either static or dynamic displays. Two-steps software was developed to analyse and translate VRML2 file format containing the object mesh and animation to a scene file with the integral imaging format. The other task is to accept texture-mapping details.

The shading modes used in the implementation are flat, Gouraud, and Phone, the lens array modes are the lenticular lens array with cylindrical lenses, and micro lens array with spherical lenses, while the projection modes are orthogonal and perspective. The modes result on an overall 9 possible implementation modes. In the case of lenticular lens array, the lenslets form an array of vertical cylindrical array, each one is a plano-convex lenslet where the back is flat and the front side is convex. Single micro lens in the micro lens array has a flat back and a spherical front side. Transformation including scaling, rotation, translation, adding colours, and animation are immediately applied to the vertices coordinates at the basic level. The normal on each triangle is calculated by implementing cross product on two of the triangle's vectors. The normal of a common vertex between neighbouring triangles is calculated by averaging the normal vectors of the adjacent triangles. The algorithm employed to calculate the normal vectors of the vertices is an original contribution as the thesis stated. The vertices normal vectors are needed to distinguish the scene vertices from the hidden ones. The resulting perimeters of the triangles in the object space are saved. A light vector was created and the light contribution for each projected point is calculated from the angle between the projectors and the light vector. Also, roughness of the surfaces, brightness, the ambient and defuse were set up and calculated for the triangles corners before projecting and shading.

The semi-cylindrical and spherical arrays give parallax in the horizontal direction and omni-directional parallax. In the micro lens case, the aperture is a square with a limited number of projection points distributed horizontally and vertically on the aperture. A spherical array gives parallax in both the horizontal and vertical directions. In the lenticular array case, the projection points are distributed horizontally on the aperture.

The finite-sized aperture models produce integral images mush faster than the integral ray tracing technique [8]. In the forward projection finite-sized aperture model, the rays are traced from the projection points to the object points. The rays are then traced from the curved surface of the lenslets to the image plane. The aim of that is calculating the exact location of the ray intersection with the surface so that the final location of the ray on the image plane is calculated and the pixel hit by each ray is located and determined. As a result, the whole image is calculated and saved to be ready for display in the replay stage. These calculations are done and the images were produced using a program written in C programming language in which different implementations were performed for each of the three groups of modes (i.e. lenticular orthoscopic, lenticular perspective, and microlens).

The methods and the necessary Equations that were used in the original thesis were modified and implemented using the application programming interface OpenGL in the environment of Visual C++ programming language. Several libraries used by OpenGL were employed to implement the projection process and the required calculations. In the original thesis, the calculations and the derived Equations aimed to tracing rays from aperture to image plane. The relation between the variables and the coordinates of the final point of the rays can be described in the form of the following function F:

F (*pos, PP, thick, curv, pitch, zv, sc, n2, shift, mode, model_mode*) = *Vf*

The output is the final location of the traced ray and the arguments are as follows:
pos = (*pos [1], pos [2], pos [3]*), the vector components of a point from the object, or the coordinates of a point from the object.
PP = (*Xpos, Ypos, Zpos*), the coordinates of a project point on the aperture.
thick is the value that measurs the thickness of the lenslet .
curv is the value that measures the radius of curvature of the lenslet.
pitch is the value that measures the pitch or the width of the lenslet.

zv the Z-coordinate of the lens array vertex, it is set to be *0*.
sc = dpi/25.4 dots per millimetre, the resolution measured as a number of pixels per mm, the resolution is selected to be equal to the normal resolution of a PC screen.
n2 the refractive index of the lenslet.

shift is the horizontal and the vertical shift *horiz_shift* and *vert_shift* the user apply to adjust the integral image on the display. These are zeroed in the examples.
mode specifies the mode of the projection, if the mode is lentecular orthoscopic, (*mode*) takes the following values: *1, 4*, and *7*.

If the mode is lentecular perspective, (*mode*) takes the following values: *2, 5,* and *8*.
If the mode is microlens, (*mode*) takes the following values: *3, 6*, and *9*.
model_mode specifies the model mode, if the model mode is *finite-sized* model, the *model_mode* takes the value *1*, if the model mode is a pinhole model, and the value of *model_mode* is *2*.

Vf the final coordinates of the intersection point between the image plane and the ray that started from the projection point and passed through the object point and the lens array *x, y,* and *z* coordinates on this point are respectively *vf [0]*, *vf [1],* and *vf [2]*.

In the original thesis, the final intersection point with the image plane for each ray was calculated as a pixel coordinates. In OpenGL, the intersection point for each ray with the image plane is calculated as a point with specific coordinates. With OpenGL functions defined and embedded in the OpenGL display program that is written and implemented in the environment of Visual C++ and supported by the libraries linked to the program such as GLUT and GLEW, OpenGL implements the projection process and provides the intensity needed for the corresponding pixel. For example, the function: *glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH)* that is used when creating top-level windows, sub-windows, and overlays to determine the OpenGL display mode for the to-be-window or overlay. In addition, some other tasks that have to be implemented within the C code in the original thesis are now implemented with OpenGL functions such as *glLightModelfv[], GLfloat lightColor[], GLfloat lightPos[],* and *glLightfv[]*. These functions are used to set the colour and position of the light and calculate the effect of the light in the scene. The functions *glTranslatef[], glRotatef[]*,and *glScalef []*

are used for translate, rotate and scale the scene points respectively. In the original thesis, all these calculations had to be done on a lower level such as multiplying and adding vectors, whereas, all these basic calculations and other tasks are included in OpenGL functions.

Figure 3.1 shows an object chosen as an example of a *3-D* object created in Blender and displayed with OpenGL. The *3-D* object is projected and the result is displayed. The mesh with its *UV* mapping is exported from Blender to an *MD2* file with the name *tris0.md2*; the texture used for *UV* mapping is saved in a file called *tris0.bmp*. *MD2* files can be imported, loaded to the *VC++* environment, and then displayed with OpenGL graphical interface.

In order to load *MD2* files, *VC++* function defined in the file called *md2_format.ccp* is used. The function *load("filename.md2")* from the class *MD2Model* is called to load an *MD2* file. This loader was modified to generate integral images as required. The header file of the loader is *md2_format.h*; this header file was often modified to suit the application as required. The file *lens_array.cpp* is the file that contains the calculations needed to project the object points on the image plane. The calculations here are based on the Equations derived by the original thesis. Projection is implemented with different modes including: lenticular orthoscopic (*1*, *4*, and *7*), lenticular perspective (*2*, *5*, and *8*), and micro lens (*3*, *6*, and *9*). In addition, two models are included: finite-sized (1), and pinhole (2). The header file of the later file is called *lens_array.h*. The file *main_OPENGL_BRUNEL.cpp* contains the main function of the program.

The program will ask the user to assign a value to *thick*, *curv*, *pitch*, the projection point coordinates *Xpos*, *Ypos*, *Zpos*, *zv*, the mode, and the model mode. In the following examples, the values are fixed as they are stated below:

Figure 3.2*: thick=3, curv=1.05, pitch=0.6, zv= -5.0, Xpos = 0.0, Ypos = 0.0, Zpos = 30, mode=1, model_mode=1.*
Figure 3.3*: thick=3, curv=1.05, pitch=0.6, zv= -5.0, Xpos = 0.0, Ypos = 0.0, Zpos = 30, mode=2, model_mode=1.*
Figure 3.4*: thick=3, curv=1.05, pitch=0.6, zv= -5.0, Xpos = 0.0, Ypos = 0.0, Zpos = 30, mode=3, model_mode=1.*

The selected model here is finite-sized. The mode is selected to be *1, 2*, and *3*. Figure 3.2 represents the orthoscopic projection of the glob on the image plane through a lentecular lens array. Figure 3.3 represents the perspective projection of the glob on the image plane through a lentecular lens array. Figure 3.4 represents the perspective projection of the glob on the image plane through a micro lens array.

The resulting images were not as they were expected. The reason is that the number of the object points are very limited, or the number of triangles are very limited, whereas, the pitch of the lenslets is much smaller than the perimeters of the triangles forming the mesh of the object.

In OpenGL the object points are projected and the positions of the projected points on the image plane are calculated for each one of these points. The resulting intersection points on the image plane (i.e. the PC screen) are connected to each other to form new triangles and these triangles are shaded and coloured with the suitable colour. The resulting images are a cluster of proportionally large triangles forming wrongly shaped objects. In order to enhance the images, a larger number of points or triangles should be employed to build the objects. If the lengths of the triangle perimeters are small so that the whole triangle's projection on the image plane can be ideally accommodated under a single lenslet, in this case, the resulting images would be shaped correctly forming the required integral images. As a result, the images would be suitable to repay with a system similar to that of the virtual capture stage.

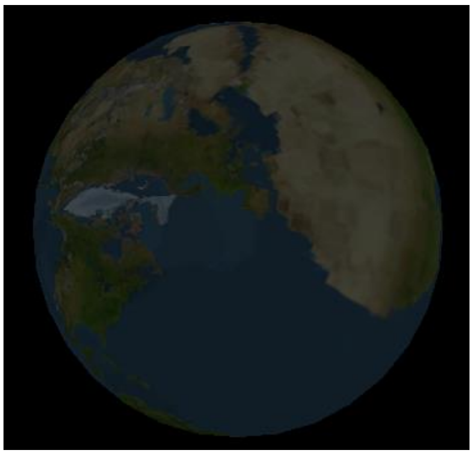


Figure 3.1: The original object.

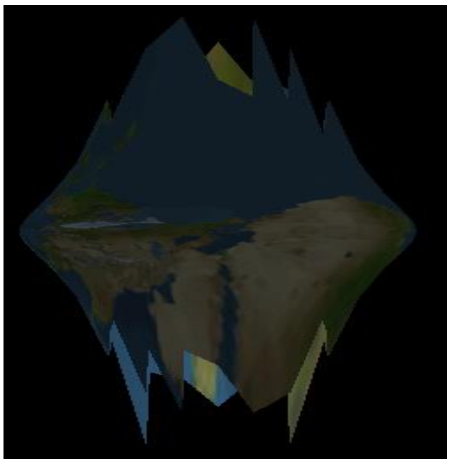


Figure 3.2: Orthoscopic/lentecular.

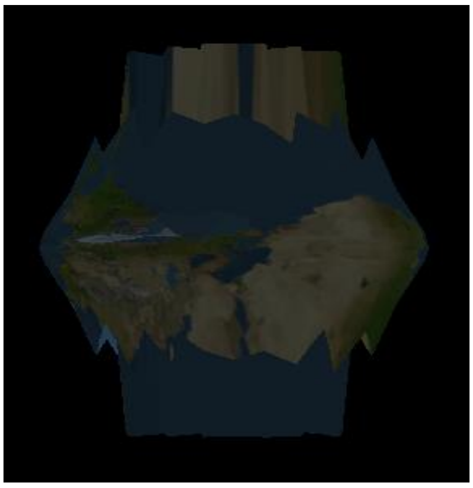


Figure 3.3: Perspective/lentecular

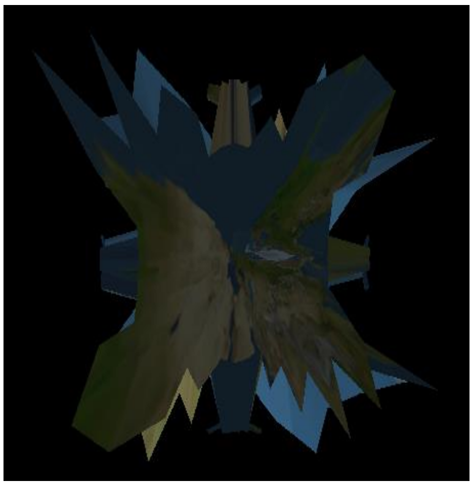


Figure 3.4: Perspective/micro lens.

3.2.2. The Forward Projection Pinhole Rendering Model

The Finite-sized aperture model as well as the pinhole model (3D-from-2D) enables perspective integral images to be produced by allowing the access of each projection point on the aperture to the whole lens array. In the Forward Projection Pinhole Rendering Model, the object points are mapped directly to the pixels so that each lenslet behaves as a pinhole camera. The rays are allowed to pass only through the vertex of each lenslet. Integral images were generated by extracting pixel intensity information from captured 2D images of a scene from different viewpoint positions. This pixel intensity information is the same as the one that is produced if the virtual lens array was present and the lenslets modelled as pinholes.

3D-from-2D image can be composed from different sub-images taken by projecting the scene from different projection points. The mode in this model is set to be lenticular perspective and both the *mode*, and the *model_mode* variables take the value of 2 in the program. Several projection points are selected from the aperture and each projection of the scene on the image plane is a sub-image. In each sub-image, the pixel columns are placed under the cylindrical lenslets in groups, each group of the pixel columns are located under a lenslet. If the number of columns in each group is $G+1$, to compose the integral image by multiplexing the sub-images, the N^{th} group of the integral image is composed from the N^{th} group of each sub-image by placing the n^{th} pixel column from the n^{th} sub-image in the $(G-n)^{th}$ location in the integral image group. In the declared approach, the pixel columns are inverted in each group, the aim of that is to reflect the fact that the ray that is passing through the pinhole intersects with the image plane in the end of the group that is opposite to the end of the projection point, otherwise the image will be pseudoscopic. For example, if the number of pixel columns is 1024, the number of the columns under each lenslet is $G = 7^{th}$. The number of groups is $N = 128$. The first group of the pixel columns is composed as follows: the 0th column of the first group of the first sub-image is placed in the $G\text{-}0 = 7^{th}$ location in the first group of the integral image. The 1st column of the first group of the first sub-image is located in the 6th location, the 7th column is located in the location 0 of the first group in the integral image, and so on.

The software used for the finite-sized model was modified to be used for the pinhole model by assigning 2 to the model mode. In addition, in order to form the integral image, a set of OpenGL and VC++ functions and scripts were employed to implement the mentioned projections and interleaving the pixel columns extracted from the different sub-images. The window size, lenslet width, the number of pixel columns under each lenslet, and the other constant parameters are selected and fixed in the program as a special case, however, these parameters can be changed either manually or after some modifications, by answering questions immediately on the screen when the program start running.

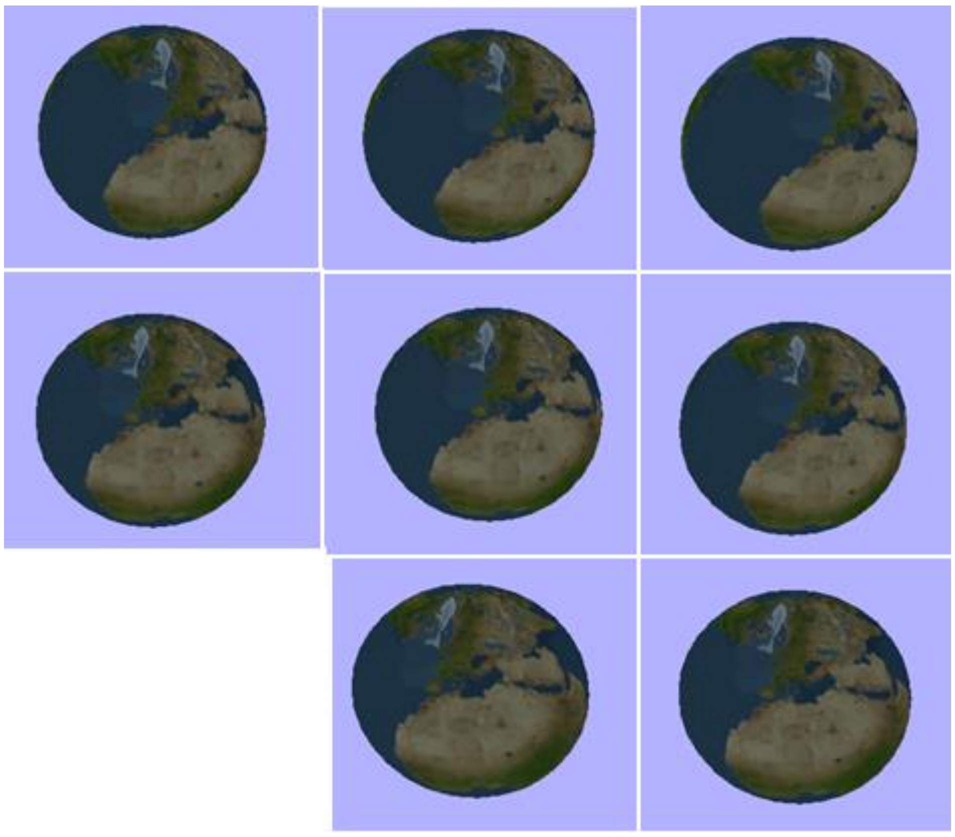


Figure 3.5: 8 sub-images of the scene projected from 8 projection points.

Figure 3.5 shows the 8 sub-images of the earth scene projected on the image plane. The first image on the right in the upper line is the projection from the left projection point on the aperture, the middle image in the upper line is the projection from the second projection point, the first image on the right in the second line is the 4th image and so on. The program projects the scene from the projection points in sequence and extracts from each resulting sub-image the pixel columns needed to form the integral image and save them. Once all the projections are performed, the information saved in memory is used to produce the integral image shown in Figure 3.6. The resulting integral image is a suitable pattern to be seen by placing a real lenticular array providing it is similar to the one used in the capture stage. The image in this case is a 3-D pseudoscopic one. To see a real scene, the pixels should be inverted as explained above.

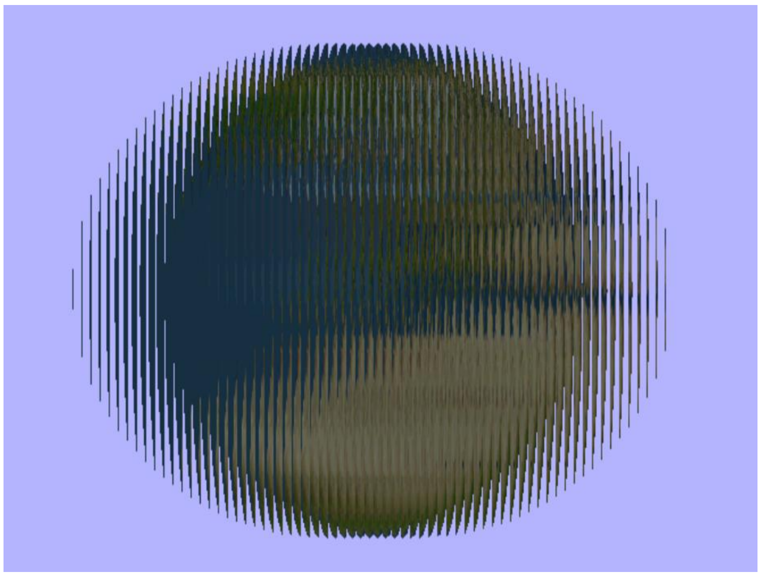


Figure 3.6: The integral image of the scene composed from the sub-images of Figure 3.5.

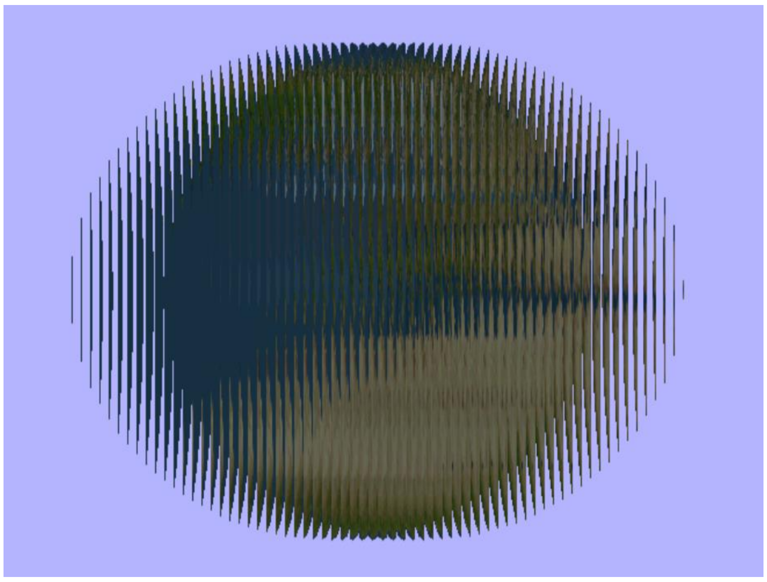


Figure 3.7 a: The integral image of Figure 3.5 with inverted pixel columns.

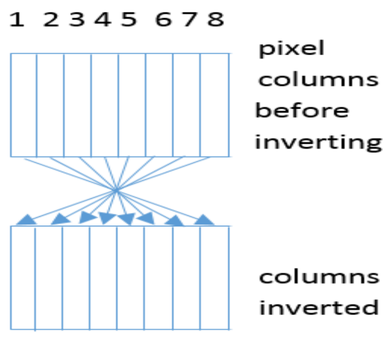


Figure 3.7 b: The method of inverting pixel columns.

Figure 3.7 shows the integral image with the pixel columns inverted around the central columns of the column groups superimposed by the lenslets. The 3-D image resulting from viewing the pattern shown in Figure 3.7 using a lenticular lens array similar to the virtual one that was used in the capture stage is expected to be a real image. The mode in both images was lenticular perspective with *mode =2*. The time needed to produce a single static image was about 2 seconds with a relatively weak PC. If we need to generate an animated integral image with this method, we need *24* images per second. Therefore, we need a more powerful machine. With that PC, each one second of the video needs 48 seconds processing time, then a 48 times faster machine is required**.**

3.3. Computer generation and rendering of Integral Images with the method of dividing image volume using OpenGL (*DIVGL*)

The suggested method is meant to be able to produce 3D static integral images and movies. The capturing stage is implemented digitally using OpenGL library in the environment of C++ programming language. The resulting 2D image is displayed on the screen of the normal PC. In order to create the integral images, different modes can be approached. The modes of creating integral images include the following:

- Perspective projection mode implemented using a cylindrical or Plano-convex cylindrical lens array with the simplification of using virtual vertical holes at the pickup stage.
- Perspective projection mode implemented using spherical lens array (micro lenses) with the simplification of using an array of pinholes at the pickup stage.
- Orthogonal projection mode implemented using cylindrical or Plano-convex cylindrical lens array with the simplification of using virtual vertical holes at the pickup stage.
- Orthogonal projection mode implemented using spherical lens array (micro lenses) with the simplification of using an array of pinholes at the pickup stage.

In this chapter, the methods based on the first two modes will be briefly discussed and some examples of the resulting integral images will be shown.

3.3.1. The method of dividing image volume

Referring to Figure 3.8, each point of the OpenGL scene is projected onto the image plane. The computer screen is now playing the role of the image plane. Practically, the image plane can be a CCD. The normal PC screen resolution is proportionally small (e.g. 90 dpi), for this reason, the elemental images that are going to be generated on the PC screen are not representing the whole scene but each elemental image represents part of the scene. For example, a 2-D elemental image with the width of eight pixels and the length of 768 pixels is created by each Plano-Convex cylindrical lenslet. Each one of the lenslets works as a separate set of pinhole cameras with an adjacent vertical set of pinholes located on the vertex of the convex surface of the lenslet. For approximation and simplifying calculations, we will consider virtual adjacent vertical barriers with vertical holes between every neighbouring two barriers. The vertical holes can be considered as a collection of pinholes. With this approximation, the collection of pinholes positioned on the vertex of the vertical lens is forming a continuous vertical opening (aperture) through which the lenslet (sees) the scene. Each lenslet can see a part of the scene through its aperture; therefore, it can only picture the part of space the lenslet can see. The accuracy of the 2-D elemental images and the quality of the produced 3-D image are proportional to the resolution of the screen on which the scene is projected. Therefore, in the case when a display device with a higher resolution is employed, a wider volume of the space can be seen by the virtual lenslet and projected on the image plane with a better approximation. Figure 3.8 shows a horizontal cross section of the projection system showing the virtual pickup stage and the real replay stage of the integral imaging process. The field of view that is seen by one of the lenslets is highlighted. The lenses can be cylindrical or spherical.

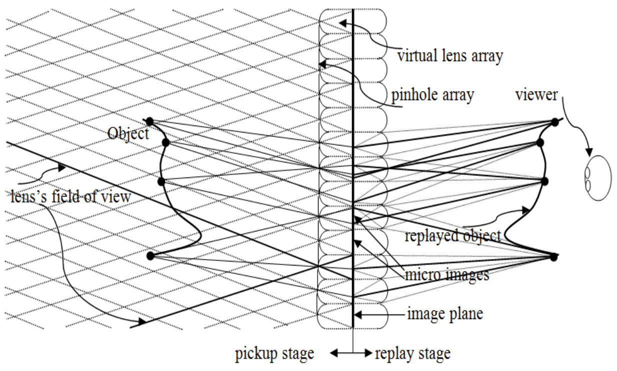


Figure 3.8: A horizontal cross section of the projection system.

Considering Figure 3.8, each lenslet images a part of the space called "frustum". Frustum is the volume that contains everything can be visible on the screen after perspective projection. The volume has the shape of a pyramid of which the apex is located at the lenslet vertex. The base of the pyramid is the far clipping plane and it is truncated at the near plane that makes the shape of a frustum. The collection of all the frustums imaged by the lenslets forms a single frustum as shown in Figure 3.9. The base of that frustum is the far clipping plane and its near clipping plane is the plane that contains the collection of viewports. The scene is projected on that plane. The image formed from the perspective projection on the plane that is containing the viewports is mapped to the image plane (CCD) so that the reverse effect of passing through the pinhole is reflected.

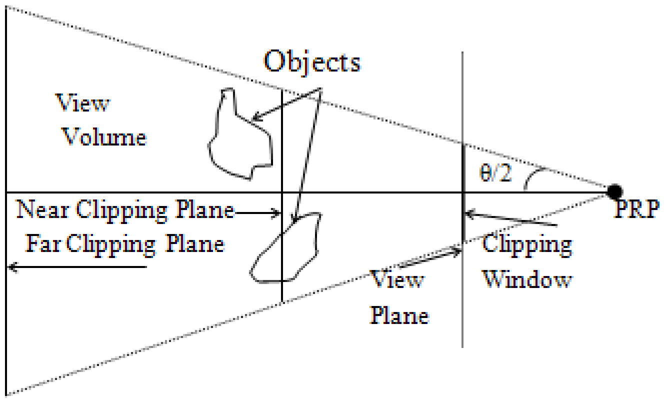


Figure 3.9: A symmetric perspective projection frustum formed from the frustums imaged by the lenslets.

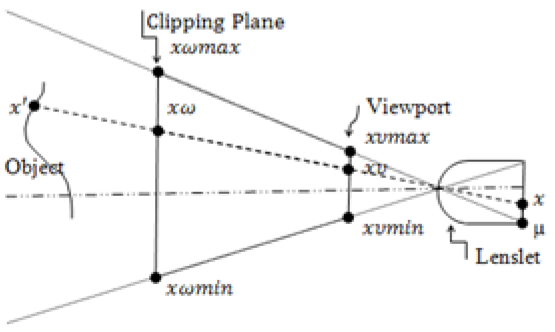


Figure 3.10: A cross section of a single lenslet with its view volume.

The Plano-Convex cylindrical lenses used in the capture stage are just virtual lenses. Cylindrical lenses have a spherical radius in one axis only, thus they magnify in just one direction, and they will change a point image into a line image [55]. The effect of the projection process is approximately similar to the effect of the real lens on the image when each lenslet changes the width of the image and compresses it in the horizontal direction without altering its height. The following two processes are identical: the perspective projection with a symmetrical frustum view volume through a viewport, and the real projection using an array of cylindrical lenses. A cross section of a single lens with the view volume of the lens, the clipping plane and the viewport is represented in Figure 3.10.

Figure 3.11 shows a horizontal cross-section of the frustums cluster and clipping planes for a few lenslets. A 3-D presentation of the lenslet and its frustum with the clipping planes is depicted in Figure 3.12.

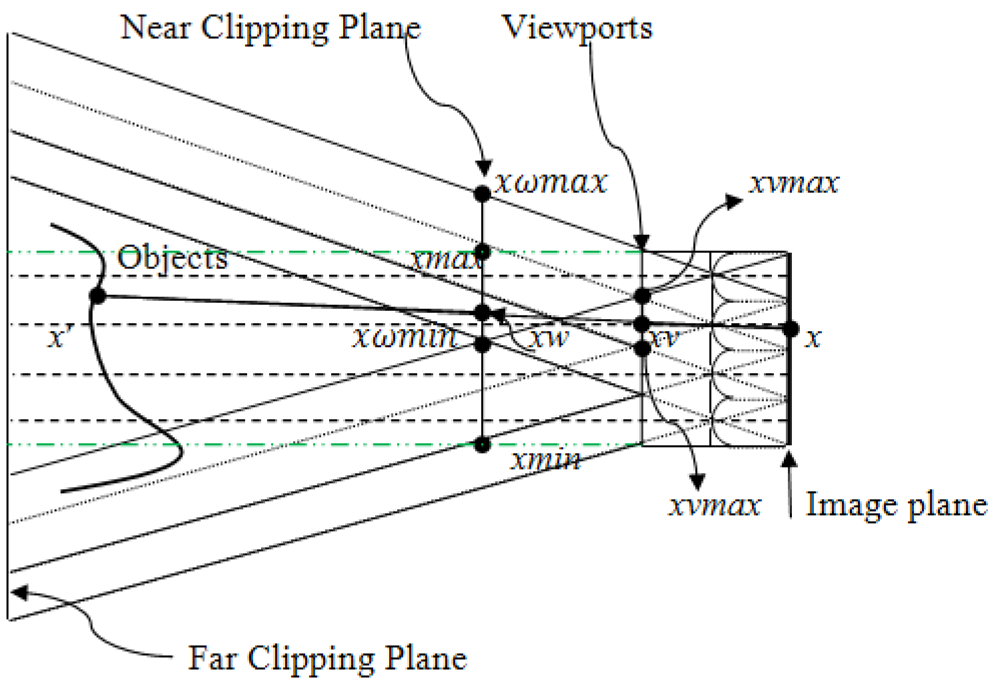


Figure 3.11: The frustums cluster for a few lenslets and the clipping planes.

The dimensions of the viewport for each frustum should be selected to be equal to the dimensions of the elemental image. In this case, the elemental image of the part of the scene that is enclosed within the symmetric frustum when it is seen through a cylindrical lens and recorded on a *CCD* film situated at the flat side of the lens, is identical to the perspective projection of that part of the scene that is projected through the viewport. When a cylindrical lens array is used, the elemental image on the view plane should be horizontally reversed, whereas, it should be reversed horizontally and vertically when a micro lens array in used.

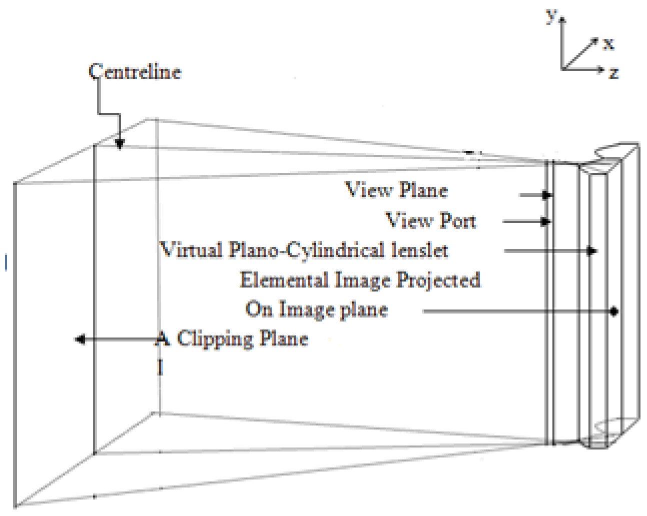


Figure 3.12: A 3-D presentation of a lenslet and its frustum.

The number of lenslets in the virtual cylindrical array is selected based on the characteristics of the display device. For example, the display device is a computer screen with *w×h pixels*. As mentioned above, the cylindrical lenses keep the height of the image and change the width as an effect of their spherical radius, for this reason, the height of the elemental image formed by the cylindrical lenslet has the same height of the image that results from perspective projection of a scene through a viewport with the same height. Thus, in order to create the elemental image for the lenslet, we need to create the equivalent perspective projection of the part of the scene. Each lenslet of the lens array creates an image of the scene from a different point of view because the cylindrical lenslets are an ensemble of vertical lenses positioned one next the other, then, each vertical lenslet has different x-coordinates in the global coordinate system. However, all the centres of the lenslets are on level with each other so that they have the same y coordinates and the same z-coordinates in the global coordinate system. At the same time, each lenslet has its own local coordinate system as it plays the role of a separate camera that produces its own elemental image on the image plane. The resulting image is a collection of elemental images situated side by side each of them is an image of a part of the scene taken from a different point of view. Each elemental image can be a compressed image of the objects contained in the lenslet frustum if we could manage to use a *CCD* with a sufficient resolution.

In our case, we aim to produce a computer image displayed on a computer screen to simulate and replace the real image that is supposed to be generated using a cylindrical lens array. In this case, we can generate elemental images so that each of them is the projection of the part of scene that is comprised within the frustum just if we use a computer screen with an adequate resolution. For example, if the width of the lenslet is *3 mm*, and the width of the screen is *120 pixels*, the whole image that needs a width of *120 pixels* to be displayed on the normal computer screen can be accommodated on the space of the elemental image when the resolution is higher than *120 pixels/3 mm = 1016 dpi*. When the resolution is lower than that, the elemental image would be approximated. In our example, the lenslet width was selected to be *3 mm* or *8 pixels* on the monitor of *90 dpi* resolution. In other words, the higher display device resolution the higher quality of the generated integral image.

When the image of the original scene is projected and displayed on the computer screen without passing the rays through a lens array or the vertical openings, the image would be accommodated on a screen with the specified dimensions w, and h. If the rays were passed through the lens array, each lenslet would compress a part of the image with specified width and height. The width and height are some of the parameters of the symmetric frustum that determine the view volume of each lenslet. For example, the height can be selected to be the same as the height of the original image but the width is selected to be $m = 8$ times higher than the width of the lenslet. In this case, if the lenslet width is *8* pixels, the width that is compressed will be equivalent to *64* pixels of the 2-D image that is supposed to be displayed without a lens array.

OpenGL will automatically apply approximation in mapping the image to the pixels when displaying the projected elemental image. Each lenslet is now considered as a camera imaging a *64 pixels* equivalent window of the image. Each lenslet will see the view volume that is shifted across the *x*-axis by a distance equals to the pitch of the lenslet. In Figure 3.13, window *5* is a window in a clipping plane that is perpendicular to the centreline of the perspective projection frustum through which the lenslet number *5* can see the view volume. Window *6* is a window in the clipping plane that is perpendicular to the centreline of the frustum of the lenslet *6*, which is shifted by the lenslet pitch from the neighbouring frustum of lenslet number *5* and so on. As an example, a virtual vertical cylindrical lens array of *128* lenslets with *8 pixels* width will be used to form an integral image with a width of *1024 pixels* and a height of *768 pixels*.

In order to specify the symmetric frustum that contains the view volume that is intended to be projected, and get the perspective projection of the part of scene within this view volume, we need to multiply the current matrix with the perspective matrix, the following OpenGL function can be used to do this task [57]:

glFrustun (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal)

Left and *right s*pecify the coordinates for the left and right vertical clipping planes. *Bottom* and *top* specify the coordinates for the bottom and top horizontal clipping planes, *nearVal* and *farVal* pecify the distances to the near and far depth clipping planes. Both distances must be positive. The values of Left, right, bottom, and top are normalized, and thus, these dimensions have the normalized values of 1, 1, 1, and 1 respectively. Bottom and top are selected to have the fixed normalized values *-1* and *+1* respectively. The maximum and minimum coordinates of the horizontal clipping planes are used to determine the bottom and top sides of the frustums; this means that the elemental images will have the maximum height which is in our example *h = 768 pixels*.

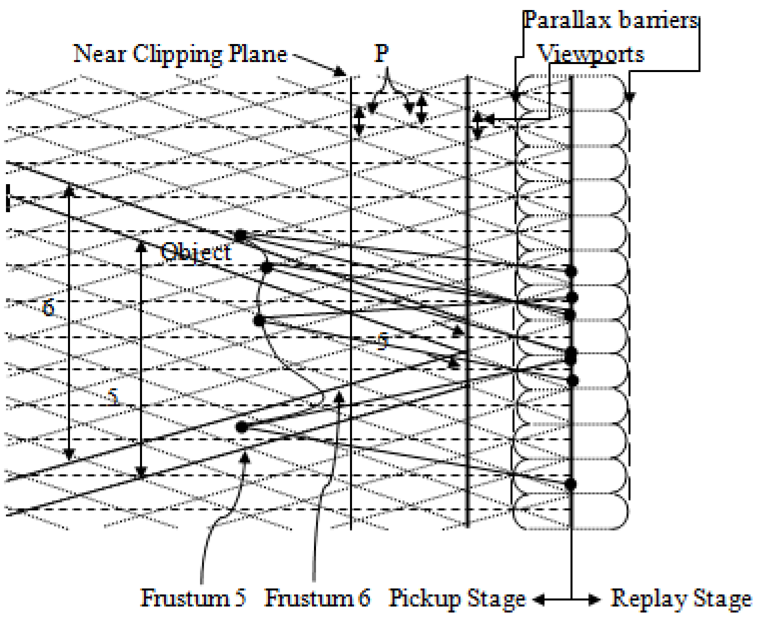


Figure 3.13: A view volume divided to frustums.

In order to calculate the left and right clipping planes, in Figure 3.13, the planes that are perpendicular to the image plane, parallel to the plane *X = 0*, and passing through the vertex of each cylindrical lenslet would be considered. Each cylindrical lenslet at the coordinate x has a plane with the Equation *X = x* that is perpendicular to the image plane and contains its vertical opening or the collection of its pinholes. If the plane *X = x* is projected on the plane *Y = 0* the projection would form the centreline of the perspective projection frustum. The projection reference point of the perspective projection frustum is located on the vertex of the lenslet. For each lenslet, the centreline is the line that passes through both the centre of the clipping window and the centre of the elemental image that is located on the image plane. For any clipping plane that is parallel to the image plane and located between the far clipping plane and the near clipping plane, the window resulting from the intersection of this plane with the frustum is actually the window that is mapped to the viewport. Through this window, the lenslet sees the scene, whereas, the scene is projected on this viewport.

The left and right parameters that are the first two arguments in the *glFrustum* function specify the virtual window at any plane parallel to the image plane and standing between the near and far clipping planes, these are normalized values rather than absolute values. Window *5* and window *6* In Figure 3.13 are examples of such windows. Each lenslet has a vertical plane containing the centreline of its frustum. Each plane of these will intersect with the clipping window through which the lenslet sees its own view volume. Therefore, there is a number of such planes equal to the number of the lenslets (*128* in our example). Each one of these planes is on a distance *P* apart from each of its neighbouring planes on the x-axis, where *P* is the pitch of the lenslet, or the width of each elemental image. If we consider any clipping plane along the view volumes of the lenslets, the clipping windows of each of two adjacent lenslets in this plane are shifted apart from each other by *P*. Then the number of the centrelines of the frustums that intersect with the clipping window is equal to the lenslets number. The width of the whole display window is equal to the entire number of *P* multiples. For example, the clipping plane at the level *8* of frustum intersections is located away from the pinholes plane by a distance equals to *8* times the focal length *F*. The clipping plane contains the clipping plane *6* with the width of $8\times P$. Levels of frustum intersections are the clipping planes that are containing the intersection lines of the vertical planes and forming the different frustums of the lenslets. For example, in Figure 3.13, the frustums *5* and *6* intersect with each other at level *1*, the intersection line between the planes forming the frustums is contained in the clipping plane at level *1*.

It was mentioned above that the 3-D image is reconstructed in the image space by the intersection of the rays produced by the micro lenses [4]. The cylindrical micro lenses replace the micro lenses that are placed at the 2-D image produced by OpenGL. Referring to Figure 3.13, the object can be seen by different lenslets, these lenslets will form images of the object points in the elemental images of each lenslet. If a lens array is used in the pickup stage and its characteristics are similar to the one's that is used in the display stage, the images of each object point are replayed and the lens array will produce rays emanating from these lenslets to intersect with each other forming a real and pseudoscopic 3-D integral image.

In the example mentioned above, each object point in the scene is supposed to be seen only by a maximum of *8* lenslets, but that is just an approximation used due to the limitation in the resolution of the display screen. Most of the object points that are situated on a distance from the lenses array exceeding the level *8* can be seen by more than *8* lenses, the near clipping plane is selected to be at a distance depending on the focal length and the lens's pitch. The part of scene that can be seen by each lens is the part that is contained in the view volume of the symmetric frustum that is defined with the function *glFrustum*.
In the case when vertical lenslets array is used, the pixel intensity values of the elemental image for each lenslet is read, saved, and then mapped to the location of pixels on the screen so that all the vertical slots are displayed at the same time forming the integral image. The location of the vertical elemental image on the computer screen is identical to the location of the corresponding lenslet in the virtual lens array. The scenes were first built in 3-D Max, or Blender applications, saved in MD2 files, and then exported to the environment of C++. Each one of the scene objects can be saved in a separated file; however, all the objects can be united in one scene and saved in one file.

The replayed 3-D scene is an image that implies on horizontal parallax. The horizontal look-around is seamless because of the fact that the cylindrical lenses are abutting each other so that the 3-D image produced by each lenslet contains many common points with the images produced by the neighbouring lenslets. In the ideal case, when the resolution of the display device is high enough, the total number of lenslets will see and image almost all the object points.

3.3.2. The capture stage

The capture stage is a simulation of the process of creating a cluster of micro images of the scene, which is processed and used to generate integral images. In order to create integral images with horizontal parallax, virtual cylindrical or Plano-convex cylindrical lens array is used. The virtual lens array superposes the image plane. The image plane is an array of pixels charge-coupled device (*CCD*) employed to record the cluster of micro images formed by the lens array. Each lens of the array captures a part of the scene and creates a micro image. To achieve full parallax, a spherical micro-lens array is used. To simplify the process, the micro-lens array is replaced with an array of pinholes. The simulation of the imaging process is based on this simplification. Each virtual micro lens (or pinhole) plays the role of a mini camera, and each micro image is a perspective projection of the scene or part of the scene that can be seen through the aperture of that mini camera.

Figure 3.8 represents a horizontal cross section of the system showing the virtual pickup stage and the real replay stage of the integral imaging process. One lens's field of view is highlighted. The lenses can be cylindrical or spherical. Each pinhole (or micro lens) in the pickup stage sees a part of the scene called the micro-lens field of view. The micro image formed by each lens is the image of the part of scene the lens sees, which is the entire part of scene that is comprised within the virtual lens's field of view. It is possible to consider the micro image formed by each virtual micro lens as a perspective projection of the points of the objects in the scene on the image plane through the lens or the aperture of the pinhole providing that the points of the objects are located in the lens's field of view. Regarding the fact that the surface of the image plane on which the micro image is recorded contains a limited number of pixels, it is possible to trace back all the rays of light received by the image plane starting from the pixels of the micro image and ending with the projected scene points. Figure 3.8 shows the incident rays that are hitting the image plane at the extreme left and right pixels of the micro images.

In order to simplify the calculations in the pickup stage, the virtual micro lens array is replaced by a virtual pinhole array. Using a pinhole array is an embodiment of the method, however, using micro lens array is supposed to produce images with higher quality. In the case of using micro lens array, the refraction of the rays should be taken into account when the pixel values are acquired and calculated. As a trade-off between the quality and the calculation complexity, a virtual pinhole array is considered instead of a micro lens array and the image pixel values resulting in this approach are approximated image pixel values supposed to be acquired when using micro lens array.

One of the tasks the application software can perform is the process of generating the integral image by creating a cluster of micro images in which the number of micro images is equal to the number of virtual spherical or cylindrical micro lenses used to capture and replay the integral image. One perspective projection is needed to create each micro image. For example, if the number of micro lenses is *100×100*, the number of projections needed is 10000. In order to reduce the number of projections and perform faster and easier rendering process of the integral images, the method of displacing the virtual camera target is introduced.

The Camera Target (*CT*) is a point in the scene space at which the camera is focusing. The Camera Central Axis (*CCA*) is the vertical axis that is perpendicular to both the lens array flat surface and the image plane. The intersection point between the flat surface and *CCA* is the Centre Point (*CP*). In this method the virtual lens array (or the virtual camera) that is utilized to capture the micro images is rotated so that *CT* is displaced, whereas *CP* is fixed. The global coordinates of *CP* are (*0, 0, and 0*). The rotation angle of *CCA* is equal to the rotation angle of the image plane and the lens array. The initial position of the lens array and image plane is the position when *CT* has zeroed *x*, and *y* coordinates, *CCA* has zeroed vertical and horizontal rotation angles, and the image formed on the image plane in this position is replaced with the Final Integral Image (*FII*).

If the micro lens array is used, the lens array and *CCA* are rotated vertically and horizontally, and *CT* is displaced to occupy different positions within the *x-y* plane, whereas, the rotation angles are only horizontal when using cylindrical lens array, and *CT* in this case is moving along the x-axis. The scene is projected orthographically on the image plane, and the resulting image is the orthogonal projection of the scene object points on the image plane. The incident rays starting from the objects points and hitting the pixels on the image plane are parallel to *CCA* and perpendicular to the image plane. Each time the micro lens array is rotated, *CT* coordinates are changed and *CCA* rotation angles are revaluated, and a new orthographic image is rendered. The technique used in this method is a process to compose a perspective integral image that is called *FII* from the different orthographic images. Specific pixels of every orthographic image are selected and mapped by value to the *FII*. The process of mapping specific pixels from the orthographic projections to the *FII* simulates the perspective projection of the scene with the same lens array. The projections of the scene object points on the *FII* that result from the mapping process are equivalent to the perspective projections of the scene object points on the initial image plane with the same micro lens array.

The camera must be positioned at the frustum central point while the target is located in the middle of the frustum clipping windows. Each point from the scene is seen from different perspectives and its projections are represented by a number of elemental images. The number of elemental images representing a single object point equals to the number of the lenslets that can see that point through either the vertical wholes, the spaces in between the barriers, or the pinholes.

Zab is the absolute value of the distance between the projection reference point and the near clipping plane. *Wab* is the width of the clipping window of the frustum. *Wab* is the width and length when micro lens array is used. *F* is the real distance between the image plane and the pinhole (or focal length) measured with the length unit. *P* is the real width of the lenticular lenslet or the diameter of the micro lens (the pitch). *Zab*, *Wab*, *F*, *P* and are linked with the following relation:

$$\frac{Wab}{Zab} = \frac{P}{F} \tag{3.1}$$

For each elemental image, the current matrix will be multiplied by the frustum matrix to produce a perspective projection of the part of the view volume that is determined by the clipping window of the frustum. The clipping window is located on the near clipping plane. The resulting matrix is multiplied by the scaling matrix that maps the clipping window of the frustum to the viewport. The width and length of the viewport are selected to be equal to the width and length respectively of the lenticular lens or the diameter of the micro lens. The locations of the viewports are selected on the screen so that the resulting elemental images are stuffed beside each other in the same way of the lens array used in the display stage. An approximation is used when projecting the scene and mapping the pixels, this approximation reduces the calculations needed to map pixels accurately on the image plane, therefore, a pinhole model is used and the effects of the lens array on the rays are neglected.

3.3.2.1. The clipping windows

If the coordinates of the clipping window are normalized, and the clipping window is comprised between the vertical clipping planes *-1, +1*, and the horizontal clipping planes *-1, +1*, the width or length of the clipping window would be *2*. The sub-windows are considered as frustum clipping windows to project the scene on the image plane and produce the corresponding elemental image for each lens. The normalized coordinates of the sub-windows can be calculated from Figures 3.8, 3.9, 3.10, 3.11, 3.12, and 3.13. The clipping window of the frustum is supposed to be located on the near clipping plane, if the lenticular lens array is used, the length of the frustum clipping windows should be the same as the length of the lenses. From Equation (3.1), the width *Wab* will be:

$$Wab \ = \frac{P}{F} \times Zab \tag{3.2}$$

Each object point in the scene is supposed to be seen by a maximum of *8* lenslets only, but that is just an approximation used because of the limitation in the resolution of the display screen. In fact, most of the object points that are situated on a distance from the lens array exceeding the level *8* can be seen by more than *8* lenses. Thus, the part of scene that can be seen by each lens is the part that is contained in the view volume of the symmetric frustum specified with the arguments of the function *glFrustum*:

glFrustum(-1.0 + p×(slot_index - 3.5), -1.0 + p×(4.5 + slot_index), -1.0, 1.0, 2.7, 200) (3.3)

The near clipping window through which the lens sees the scene is defined with the first *4* arguments. If each clipping window has the width of *8×8 pixels*, the whole clipping window for the lens array has the same dimensions of the display screen (i.e. the computer screen). The two dimensions (i.e. height and width) of the clipping window of the array are normalized so that the clipping planes have coordinate positions at *xmin = -1, xmax = +1, ymin = -1, and ymax = +1*. *slot_index* is the *x*-coordinate of the frustum centreline (in our case the frustum centre plane). The first frustum centerplate of the first lenslet is located on the same *x*-coordinate of the vertex of the first lenslet that is equal to *P/2*. The second centre plane coordinate coordinate is *slot_index* = *1.5*× *P* and so on.

Based on the previous assumptions, and referring to Figure 3.11, the whole screen or the whole clipping window is equal in width to the width of the *128* lenslets, then the normalised width of a single lenslet is equal to *P = 2/128 = 1/64 = 0.015625* of the normalised width. For a clipping window with the width of *8×P*, the clipping window will be located at the level *8* as it is depicted in Figure 3.11. The first lenslet in the far left is given the number *0*.

Starting with the lenslet number *4* with the *slot_index = 4*, the normalized width of the clipping window is $\Delta = 8\times P = 8\times 0.015625 = 0.125$. the left clipping plane of the 4th lenslet has the normalized coordinate of $-1+ P/2 = -1 + P\times(4 - 3.5) = -1 + 0.0625$, and the right clipping plane coordinate = $-1 + P/2 + 8\times P = -1 + P\times(4.5 + 4)$. In general, the left and the right coordinates of the clipping windows for the lenslets between *4* and *124* are respectively as follows:

X_left = *-1.0 + p×(slot_index - 3.5)* (3.4)

X_right = *-1 + p×(4.5 + slot_index)* (3.5)

The view volumes of the first *4* lenslets are partially contained in the scene view volume, and the width of the clipping window for each lenslet is less than *8×P*. For those lenslets, the variable *slot_index* is less than *4*, and the function *glFrustum* takes the arguments: *-1* as the normalised *x*-coordinate of the left vertical clipping plane and *-1 + p× (4.5 + slot_index)* for the right vertical clipping plane, *-1* and *+1* for the top and bottom clipping planes:

glFrustum(-1.0, -1 + p×(4.5 + slot_index), -1.0, 1.0, 2.7, 200) (3.6)

The arguments of *glFrustum* function for the last *4* lenslets are as follows:

glFrustum(-1.0 + p×(slot_index - 3.5), 1.0, -1.0, 1.0, 2.7, 200) (3.7)

3.3.2.2. The analogy between the lens effect and the viewport mapping in OpenGL

Referring to Figures 3.9, 3.10, and 3.11, in order to state the analogy between the lens effect and the viewport mapping in OpenGL, we need to know that that object descriptions are transferred to the viewport using the transformation that maintains the same relative placement of a point in the viewport as it had in the clipping window [5] [56]. In order to transfer the global coordinate point into the same relative position within the viewport, we require that:

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{x\omega - x\omega_{min}}{x\omega_{max} - x\omega_{min}} \qquad (3.8)$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{y\omega - y\omega_{min}}{y\omega_{max} - y\omega_{min}} \qquad (3.9)$$

where $x\omega, y\omega$ is the position in the clipping window that is mapped into position xv, yv in the associated viewport. xv_{min}, yv_{min} $, xv_{max},$ and $yv_{max},$ are the coordinates of the lower left corner and upper right corner of the viewport respectively, whereas, $x\omega_{min}, y\omega_{min}$ $, x\omega_{max},$ and $y\omega_{max},$ are the coordinates of the lower left corner and upper right corner of the clipping window respectively.

On the other hand, the effect of the vertical Plano-convex lenslet on the image is almost similar to the window-to-viewport transformation process that is implemented in OpenGL. Figure 3.10 illustrates the projection of an object point through a vertical lenslet; the refraction that is expected to occur is such lenslet is neglected. In addition, the vertex of the lenslet is considered as the focal point of the lenslet. These approximations make the image rendering process easier and faster, however, more accuracy can be achieved with considering all these details, but the price is a longer rendering time, and expensive calculations. Figure 3.10 shows the intersection between the frustum of a lenslet and the plane that is parallel to the plane X-Z and perpendicular to the Y-axis. The point x' is an objet point located in the view volume of the lenslet, and x is its projection on the image plane. From the similarity between the triangles, and Thales formulas we can write:

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{x\omega - x\omega_{min}}{x\omega_{max} - x\omega_{min}} \qquad (3.10)$$

Equation (3.10) is identical to Equation (3.8) that is implemented in OpenGL through the transformation process. The length of the viewport was chosen to match the length of the clipping window, and the vertical Plano-convex cylindrical lenses do not magnify in the y direction. Thus, for a position $x\omega, y\omega$ in the clipping window that is mapped into positionxv, yv, we have:

$$yv = y\omega \qquad (3.11)$$

$$yv_{min} = y\omega_{min} \qquad (3.12)$$

$$yv_{max} = y\omega_{max} \qquad (3.13)$$

From Equations (3.10), (3.11), (3.12), and (3.13), we can write:

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{y\omega - y\omega_{min}}{y\omega_{max} - y\omega_{min}} \qquad (3.14)$$

Equation (3.14) is identical to Equation (3.9) that is implemented in OpenGL through the transformation process. From Figure 3.10, the point xv_{max} in the viewport is mapped to the point μ in the image plane; also, the point xv is mapped to the point x that is the final projection of the point x' and so on. The final projection points of the scene on the image plane can be obtained by flipping the points in the viewport so that their positions are reversed around the central vertical axis of the viewport. This is implemented in OpenGL by mapping the pixels intensity values in the elemental image to their symmetrical pixels in the opposite side with equal distances from the centre of the elemental image. The coordinates of the vertical planes of the clipping window are identical to the *x-coordinates* of the clipping window (i.e.$(x\omega_{min})$, and $(x\omega_{max})$), whereas, the horizontal clipping planes will be *-1*, and *+1* in the case of using lenticular lenses and $(y\omega_{min})$, and$(y\omega_{max})$ in the case of using micro lenses. The point at the apex of the lens or the pinhole is considered as a location of a camera pointing on the middle of the symmetric frustum that contains the view volume that holds the part of the scene supposed to be imaged by this camera. From the relation (3.8), any point in the scene such as $x^{'}$ contained in the view volume will be projected in reference to the projection reference point. The reference point is the location of the camera and it is the intersection point between the vertical hole located in between the two horizontal barriers of the lenslet and the horizontal plane that contains the object point. The projection of point $x^{'}$ in the frustum's clipping window is $(x\omega)$. In the OpenGL application interface, the function *glFrustum* describes the perspective matrix that is used to produce perspective projection. The perspective matrix is multiplied by the current matrix and the resulting matrix replaces the current matrix [57]. The perspective matrix is:

$$\begin{matrix} \frac{2Zn}{(x\omega_{max} - x\omega_{min})} & 0 & \frac{(x\omega_{max} + x\omega_{min})}{(x\omega_{max} - x\omega_{\min})} & 0 \\ 0 & \frac{2Zn}{2} & 0 & 0 \\ 0 & 0 & -\frac{(Zf + Zn)}{(Zf - Zn)} & \frac{(2Zf.Zn)}{(Zf - Zn)} \\ 0 & 0 & -1 & 0 \end{matrix}$$

Zf and *Zn* are the z-coordinates of the far and near clipping planes respectively. With approximation, we will suppose that the clipping window is equal in size to the viewport or the effective surface of the image plane where the total integral image content is displayed. The value of Zab can be selected in a way that each point in the scene is projected to two elemental images at least. Each object point in the scene will be replayed by at least two lenses, each projection of these should be done from different perspective so that the rays starting from the point projections in different elemental images are intersected in a real or virtual point to from the image of that object point. The selected minimum number of projections is dependent on the available resolution of the screen, the higher resolution of the screen the larger width of the clipping window that can be accommodated in the elemental image, and therefore, the higher number of projections can be taken for the same object point. If the clipping window of the frustum was selected to be located on the viewport, the number of projections for each object in the scene is *1* and the number of elemental images that contain the projection of each point is *1*. The number of elemental images that is enough for our purpose is *2* at least. To achieve that, we need to increase *Zab* to be able to include projections of points in multiple elemental images.

If $\frac{P}{F}$ = 2/3, and the number of projections for each point is selected to be *m*, then $\frac{m\times P}{m\times F}$ = 2/3, in this case *Zab* = *m*×*F*. *Zab* is identical to the selected distance between the projection reference point and clipping window on the near clipping plane. The width of this clipping window is supposed to be equal to the width of the image plane that is equal to $n\times P$ where *P* is the pitch of the lenslets or the diameter of micro lens, and n is the number of lenslets or micro lens in *x* or *y* direction. Therefore, each object point in the scene is supposed to be seen by a maximum of *m* lenslets; this number is selected in dependence to the limitation of the display screen resolution. The part of scene that can be seen by each lens is the part of the scene that is contained in the view volume of the symmetric frustum defined with the OpenGL function *glFrustum*:

glFrustum(left, right, bottom, top, zNear, zFar) (3.15)

where *left* is the coordinate of the left vertical clipping plane, *right* is the coordinate of the right vertical clipping plane, *bottom* is the coordinate of the bottom horizontal clipping plane, *top* is the coordinate of the top horizontal clipping plane, *zNear* is the distance to the near-depth clipping plane, *zFar* is the distance to the far-depth clipping plane. The near clipping window through which the lens sees the scene is defined with the first *4* parameters. When the cylindrical array is used, each elemental image has the width of *N*×*p*. *N* is the number of horizontal pixels in the elemental image, and p is the width of the pixel. m is the required projections for each object point in the scene, the frustum clipping window has the width of $N \times p \times m$. The total number of lenslets is n, the whole clipping window for the lens array has the same dimensions of the display screen, the two dimensions (height and width) of the clipping window of the array are normalized so that the clipping planes have coordinate positions at $x_{min} = -1$, $x_{max} = +1$, $y_{min} = -1$, and $y_{max} = +1$. *lenslet_index* is the normalized x coordinate of the frustum centreline (in our case the frustum centerplane). The left first frustum centerplane of the first lenslet is located on the same distance from *xmin* = *-1* of that of the first lenslet vertex on left (i.e. *P/2*). The distance of the second centerplane to the left edge is $1.5 \times P$ and so on. Based on the previous assumptions, the normalised width corresponding to the width of a single lenslet is equal to $p = 2/n$ of the normalised width. The normalized width of the frustum clipping window is $m \times 2/n$. The first lenslet in the far left is given the number *0*, if *zFar* is chosen so that the frustums are enough to contain the whole scene, and *zNear* is calculated from $Wab = \frac{P}{F} \times Zab \times N \times p \times m = \frac{P}{F} \times zNear$, then:

$$zNear = N \times p \times m \times \frac{F}{P} = N \times p \times m \times \frac{F}{N \times p} = m \times F \qquad (3.16)$$

Then the OpenGL function *glFrustum* for the first *m/2* lenslets will be:

glFrustum(-1.0, -1 + p×(m/2+ 0.5 + lenslet_index), -1.0, 1.0, zNear, zFar) (3.17)

For the lenslets from number *m/2* upward to *n – (m/2)*:

glFrustum(-1.0 + p×(lenslet_index - 0.5 + m/2), -1.0 + p×(m/2+ 0.5 + lenslet_index), -1.0, 1.0, zNear, zFar), (3.18)

For the lenslets from number *n – (m/2)+ 1* upward to *n-1*:

glFrustum(-1.0 + p×(lenslet_index - 0.5 + m/2), 1.0, -1.0, 1.0, zNear, zFar) (3.19)

3.3.2.3. The resulting integral images

The pixels intensity values of the elemental image for each lenslet are read, saved, and then mapped to pixel locations on the screen so that all the vertical slots are displayed at the same time forming the integral image. The location of the vertical elemental image on the computer screen is identical to the location of the corresponding lenslet in the virtual lens array. Figure 3.14 shows a simple 3-D scene “cutlery” rendered with OpenGL. The scene was first built in 3D max, and then exported by OpenGL via Blender and saved in MD2 files. Each object in the scene was exported, saved and used as a separate MD2 file. Figure 3.15 shows the integral image resulting from the projection of the scene through the virtual lens array.

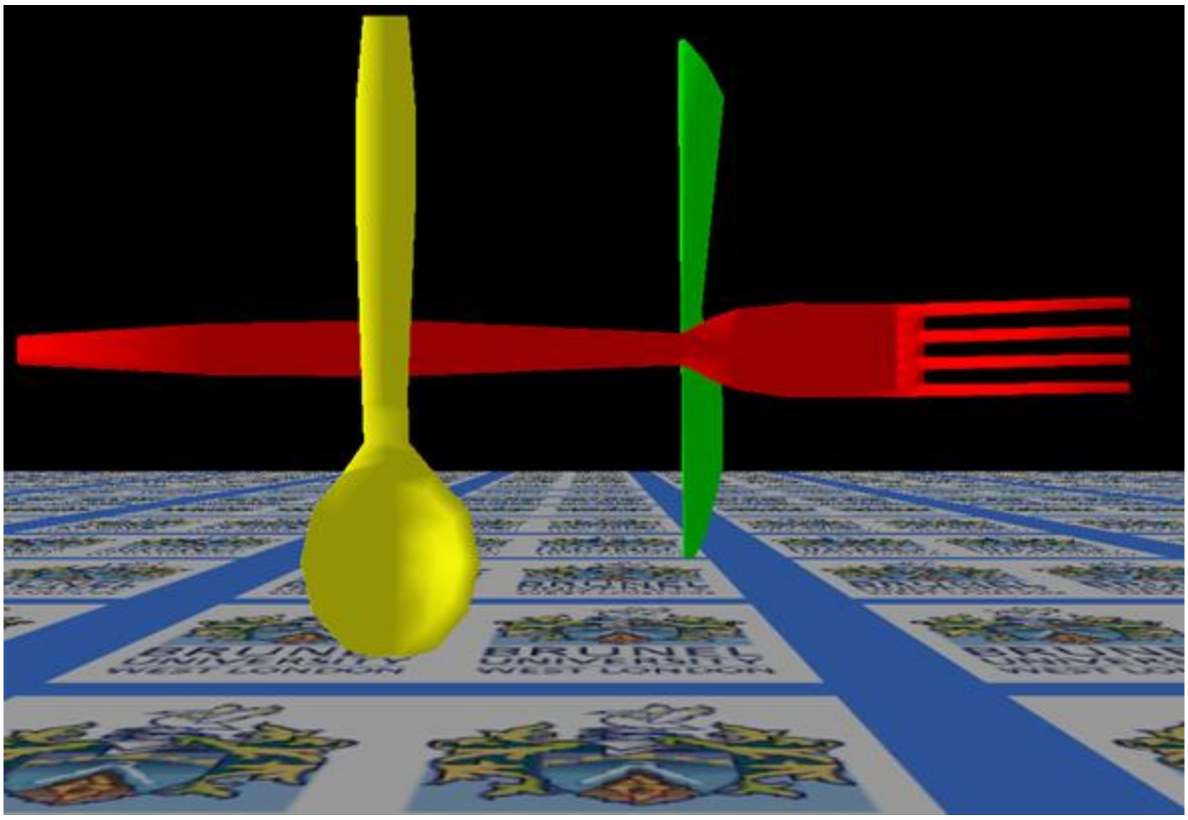


Figure 3.14: An OpenGL 2-D simple scene.

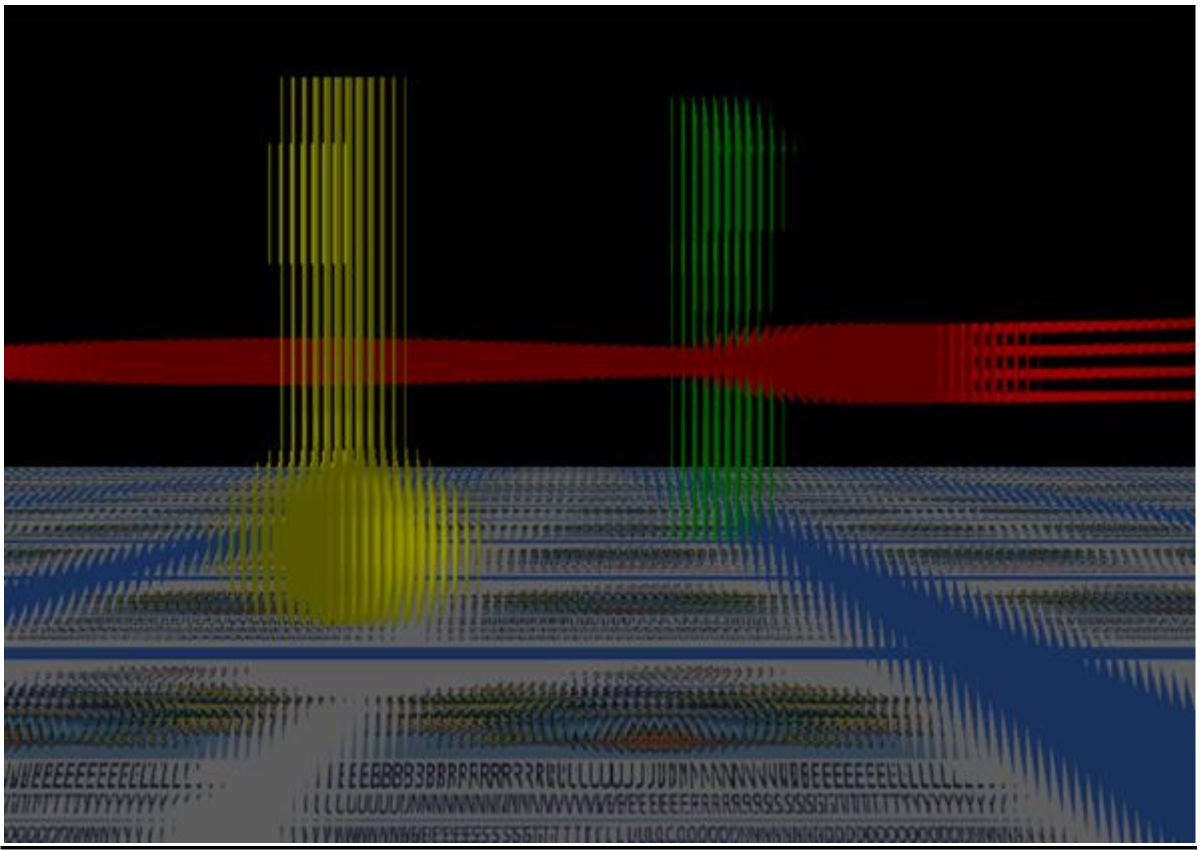


Figure 3.15: The integral image resulting from the projection of the simple scene.

To define a clipping rectangle using OpenGL, the following function can be used:

void glScissor(GLint x, GLint y, GLsizei width, GLsizei height)

x and *y* are the coordinates of the bottom left corner of the rectangle width and height are self-explanatory. The simplified method is the case when the scene is projected on the image plane through a virtual array of vertical holes (see Figure 3.12). The replayed 3-D scene implies on horizontal parallax and the horizontal look-around is seamless because of the fact that the cylindrical lenses are abutting each other so that the image produced by each lenslet contains many common points with the image produced by the neighbouring lenslet. The volumetric parts of the scene are intersected. The intersection can happen when at least two lenslets replay the same point of the scene. When the same point from the scene is projected on at least two different micro images, the replayed images can intersect and form a real image in front of the display screen.

In the ideal case when the resolution of the display device is high enough, every lenslet from the lens array can see all the object points, and then .every micro image contains information about any object point in the scene. Due to the analogy between the real and the simulated cylindrical lens array, the rendered micro images are almost identical to the micro images achieved by the real cylindrical lens array.

In computer graphics, the cone of vision is approximated with a symmetric frustum, and we can use a field-of-view angle to specify an angular size for the frustum [5]. Using the viewport, we can control the position within the display window. The clipping window selects what we want to see, the viewport indicates where it is to be viewed on the output device [5]. In order to display a 3-D scene on a display device, the graphics packages perform normalization, clipping routine, and viewport transformation. Object proportions are maintained if we set the aspect ratio of the viewport area to the same as the clipping window.

Figure 3.16 shows a 2-D image created in Blender, exported to C++ & OpenGL environment, converted and displayed as a 3-D autostereoscopic integral image using the method explained above. Figure 3.17 is the 3-D image for the 2-D image shown in Figure 3.16 when a lenticular lens array is used.

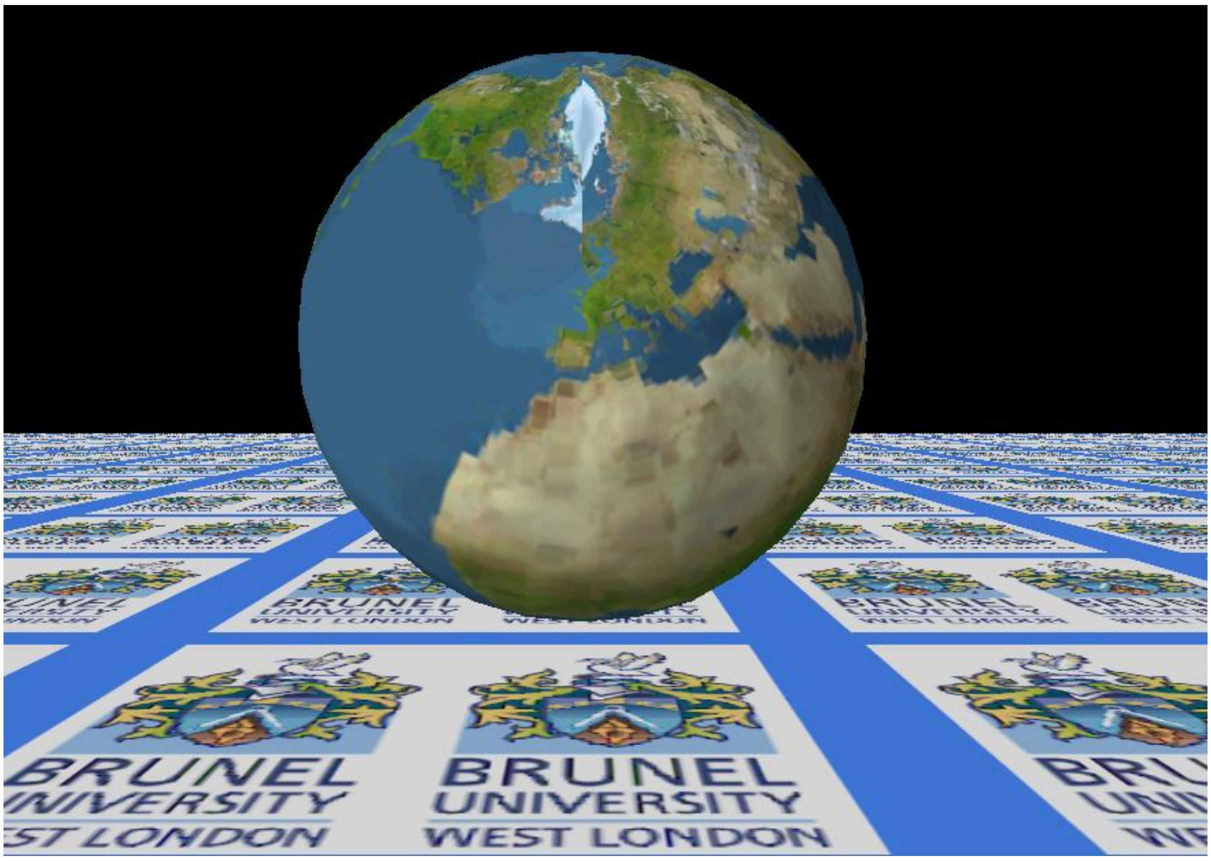


Figure 3.16: A 2-D image to be converted and displayed as a 3-D integral image.

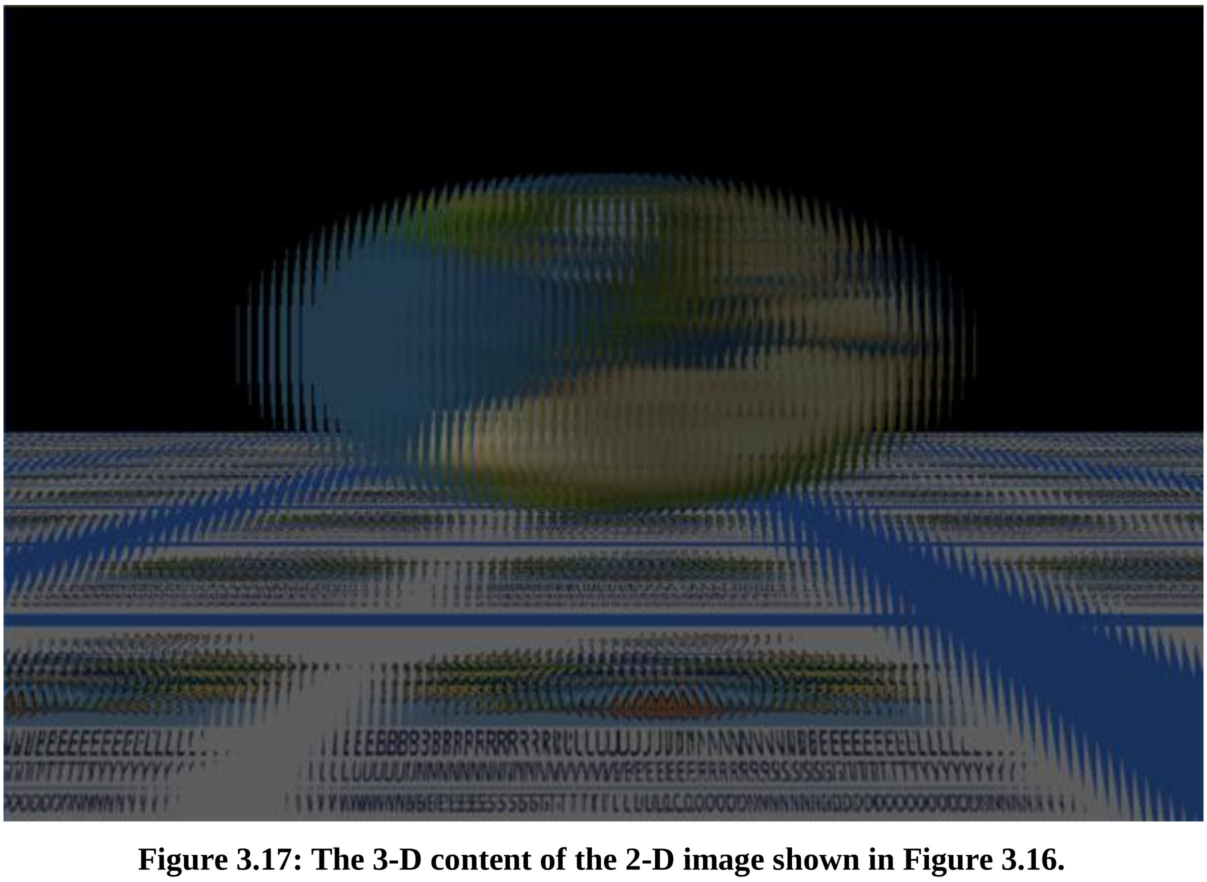


Figure 3.17: The 3-D content of the 2-D image shown in Figure 3.16.

3.3.3. The replay stage

The lens array used to replay the image is supposed to have the same characteristics and parameters of the virtual lens array used to capture the image. If the pixels of the micro images in the viewport are mapped to the image plane without changing their positions in proportion to each other, replaying the image produces an orthogonal virtual image. That is because the rays emitted from the lenses in the replay stage that are belonging to the same object point are in this case parallel to the projection rays of the same point in the pickup stage. Therefore, the rays belonging to the same point intersect in a virtual point behind the screen. As a result, the viewer will see an autostereoscopic orthogonal virtual image located behind the screen, as it is depicted in Figure 3.18.

Alternatively, when the pixels of the elemental images in the viewport are rotated around the centre of the elemental image in the case of microlense or flipped around the vertical central column of the elemental image in the case of lenticular lens array, a pseudoscopic real image will appear to the viewer. That is because the rays emitted from the lenses in the replay stage

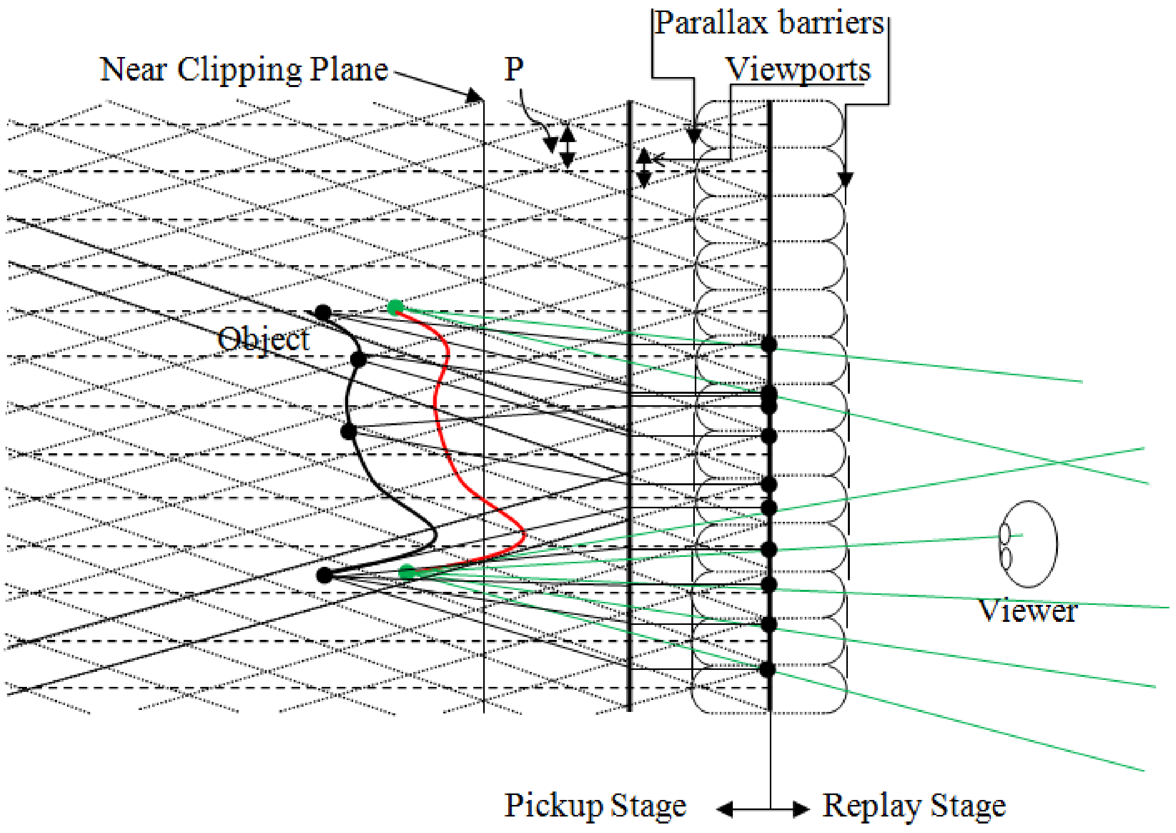


Figure 3.18: The pickup and replay stages of an orthogonal virtual integral image.

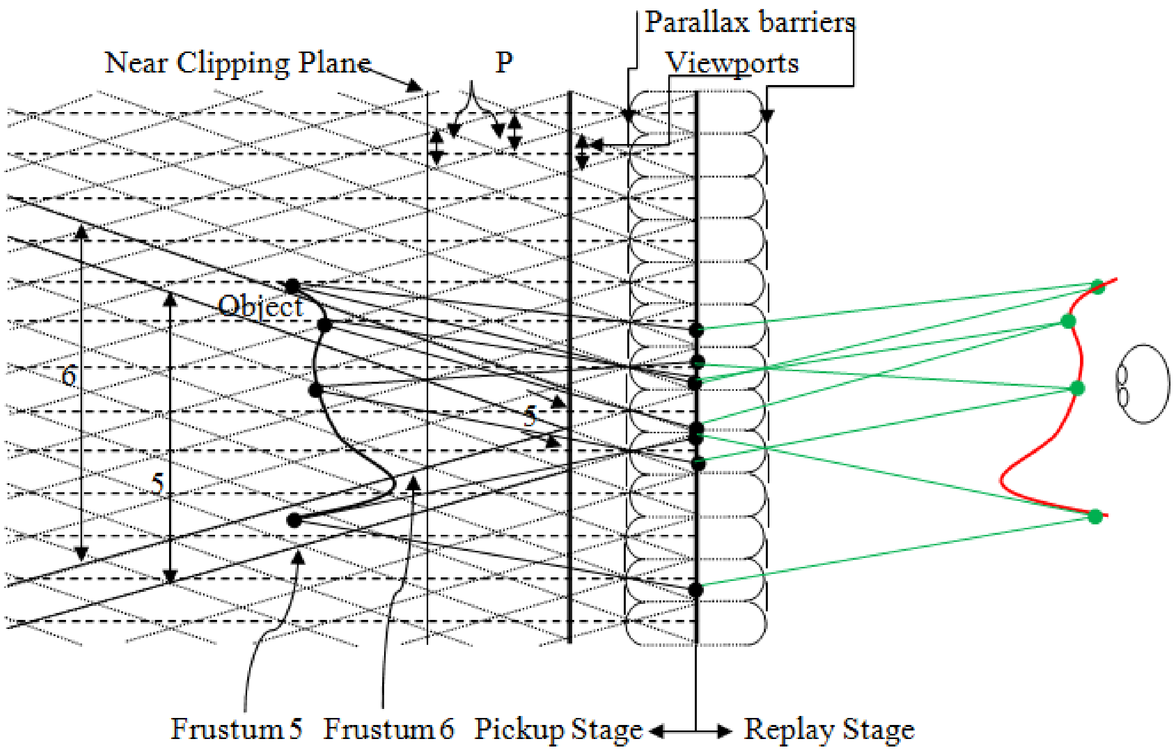


Figure 3.19: The pickup and replay stages of a pseudoscopic real integral image.

that are belonging to the same object point are in this case flipped horizontally in comparison to these of the same point in the pickup stage. Therefore, these rays of the same point intersect in a real point in front of the screen. As a result, the viewer will see an autostereoscopic pseudoscopic real image located in front of the screen, as it is depicted in Figure 3.19.

3.3.4. Perspective projection using spherical lens array:

In order to simulate the elemental image formed by a spherical lens, the viewport now is set to be rectangular with a width and a height equal to the diameter of the proposed lens array. The spherical microlenses can be arranged in a way so that the microlenses are vertically aligned in every other row, whereas, the microlenses in a row are shifted to the microlenses in the next row with a distance that is equal to the microlens radius as it is depicted in Figure 3.20.

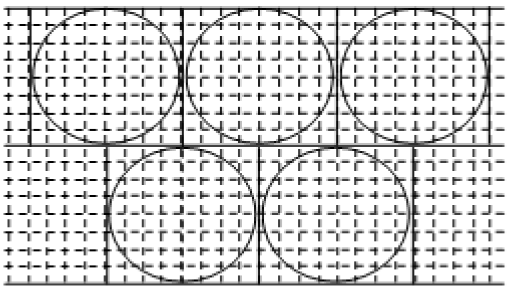


Figure 3.20: Spherical microlenses and rectangular viewports of *8×8* pixels.

The same discussion for the case of the cylindrical lens array is applicable for the case of the spherical lens array. There are few differences between the two cases including the need for a vertical scanning in addition to the horizontal scanning in the case of spherical lens array.

In the following example, the number of the elemental images in the horizontal direction is *128,* and in the vertical direction *128*. The screen is *1024×768 pixels*; therefore each elemental image is *8×6 pixels*. In this example, the proportion between the width and height of the elemental image is equal to that of the whole image displayed on the PC screen. The number of the elemental images that are supposed to be formed is *128×128 = 16384*. This number is very high and this will cause the system to crash if the memory is insufficient.

The excessive projections of the scene can cause the system to run out of memory in the case that the size of the video card memory, or the system memory are insufficient to meet the requirements of the projection process, for this reason, when the projection number reaches a specific stage the system crashes. Figure 3.21 shows the image formed on the image plane if the scene that is shown in Figure 3.16 was projected through a microlenses array of *128×128* microlenses with the dimensions *8×6* pixels. Figure 3.22 shows the image that can be placed behind a spherical microlenses array in which the diameter of a lens is equal to the length of the elemental image side. Figure 3.23 is the image when the elemental image is *16×16* pixels. In real life, the elemental images are pseudoscopic after passing through the lens array. In order to simulate the effect of the lens on the image, the elemental images are inverted around the centre in Figure 3.23. The case shown in Figure 3.24 is the one when the micro lenses of each row in the microlens array are aligned vertically with the micro lenses in the other rows (i.e. an orthogonal microlens array). The dimensions of the elemental images are *8×6* pixels each.

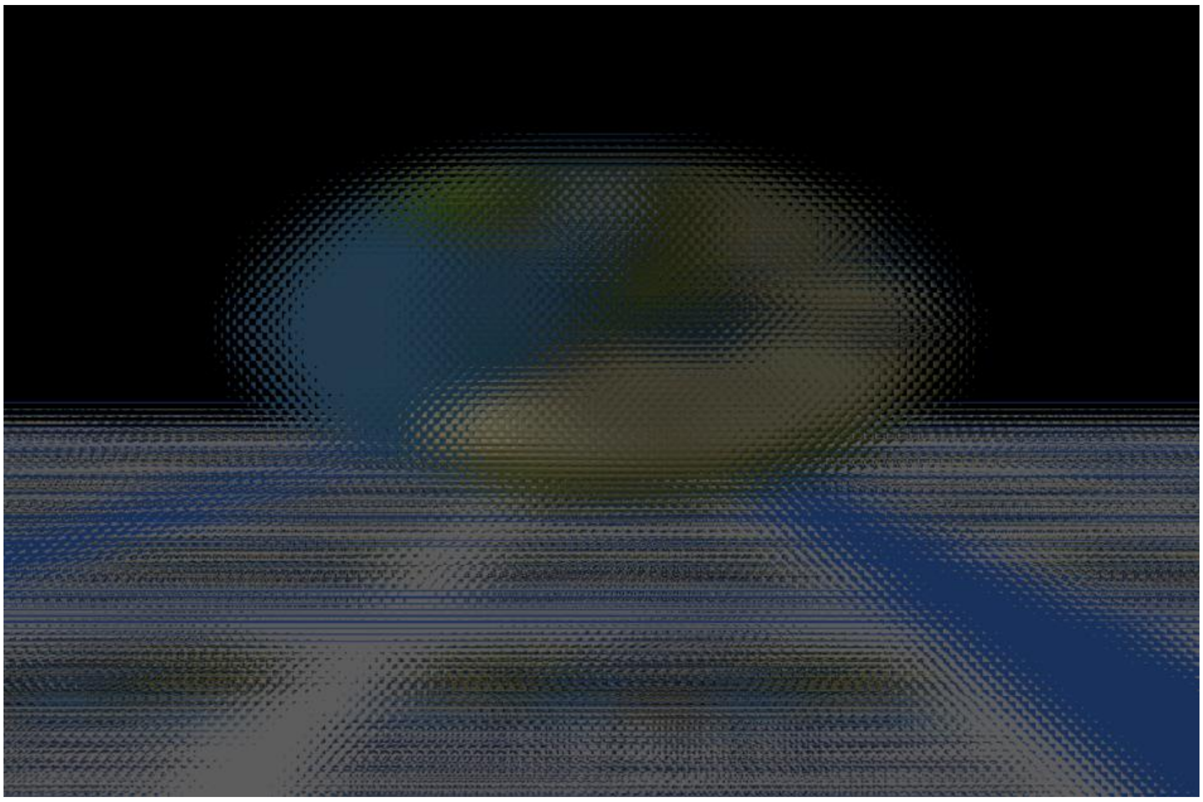


Figure 3.21: An image formed by a microlens array of *128×128* microlenses with *8×6* pixels each.

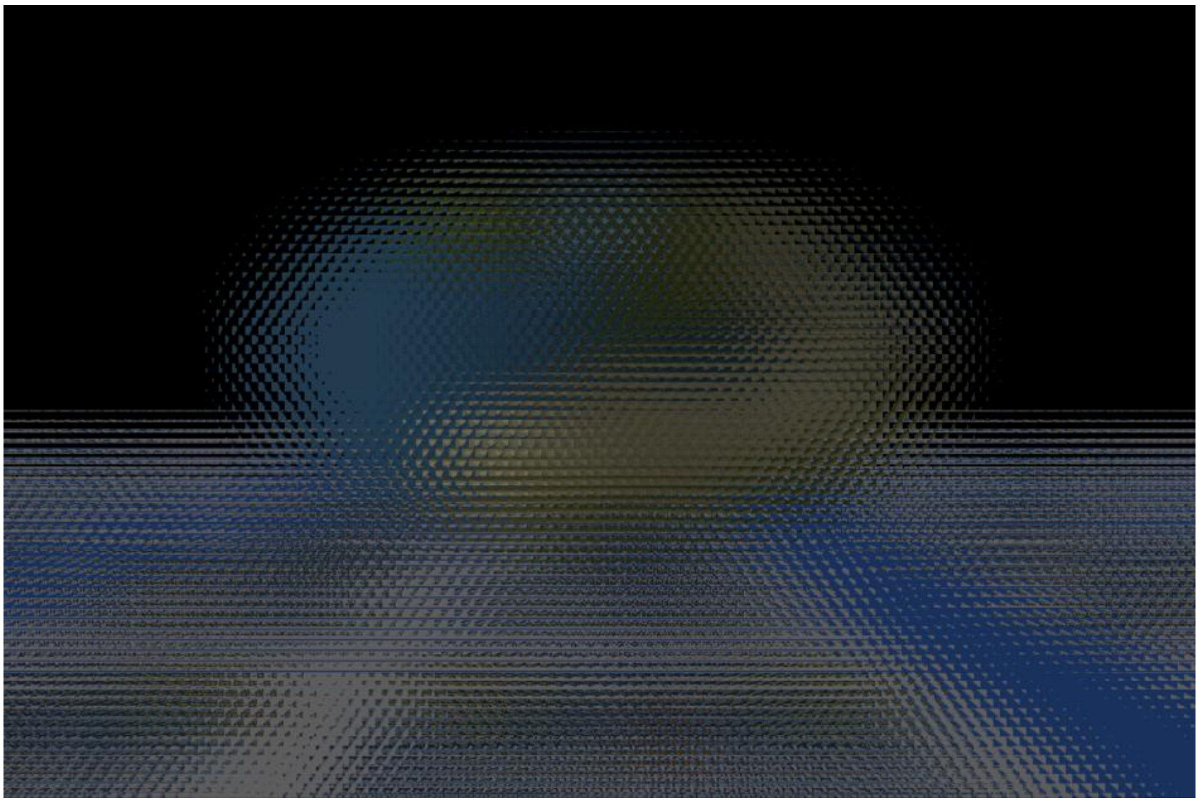


Figure 3.22: An image formed by a microlens array of *128×96* microlenses with *8×8* pixels each.

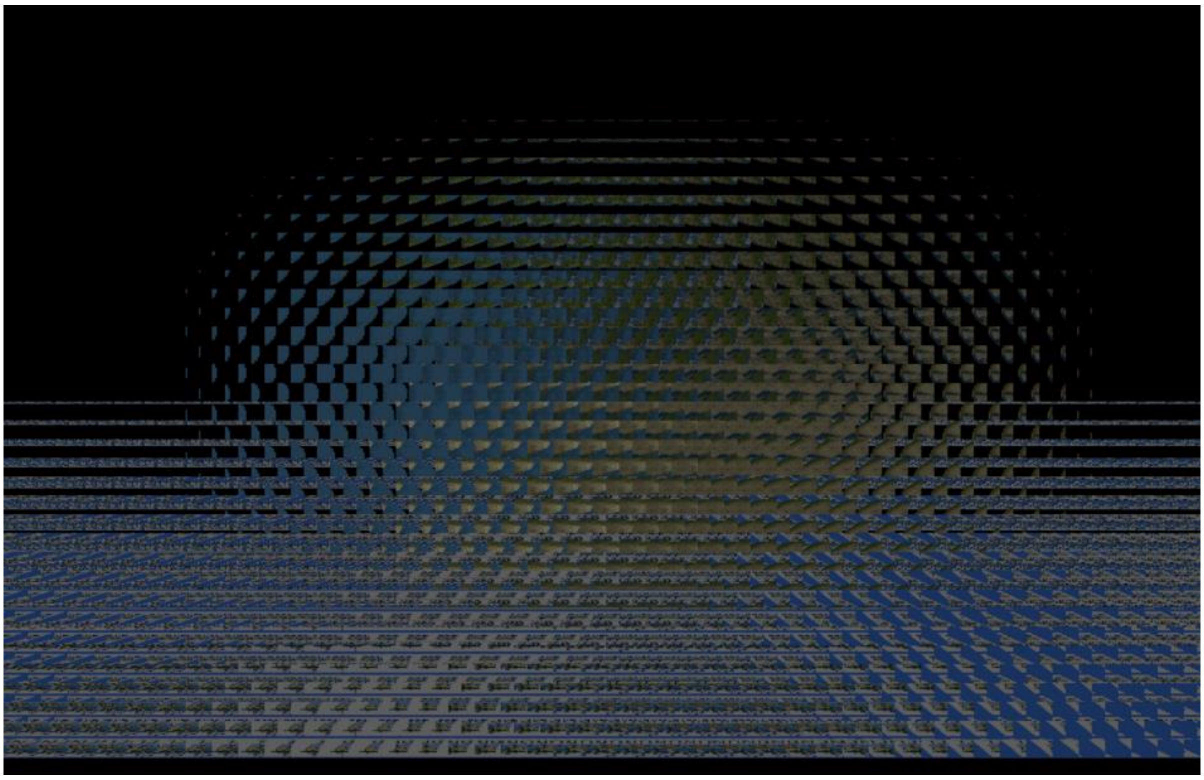


Figure 3.23: The image formed by a microlens array of *64×64* microlens with *16×16* pixels, the elemental images are inverted around the centre.

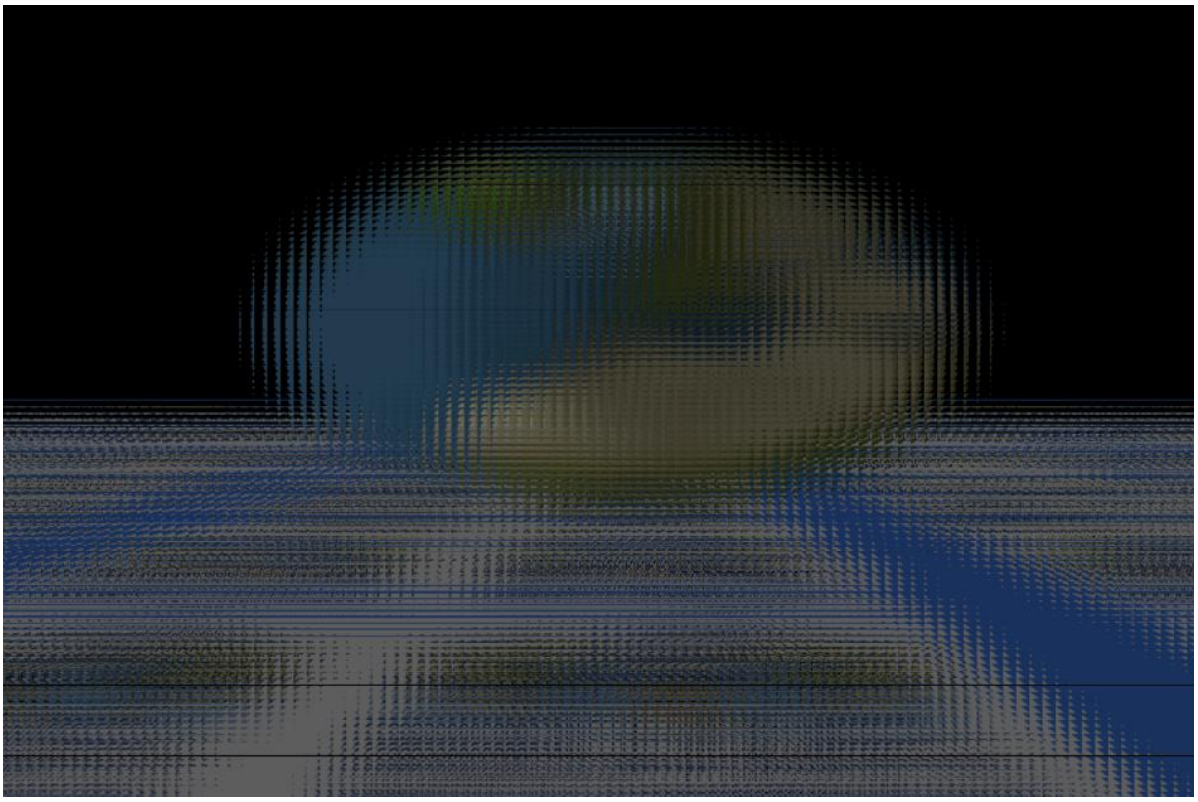


Figure 3.24: The image formed by an orthogonal microlens array of *128×128* microlens with *8×6* pixels.

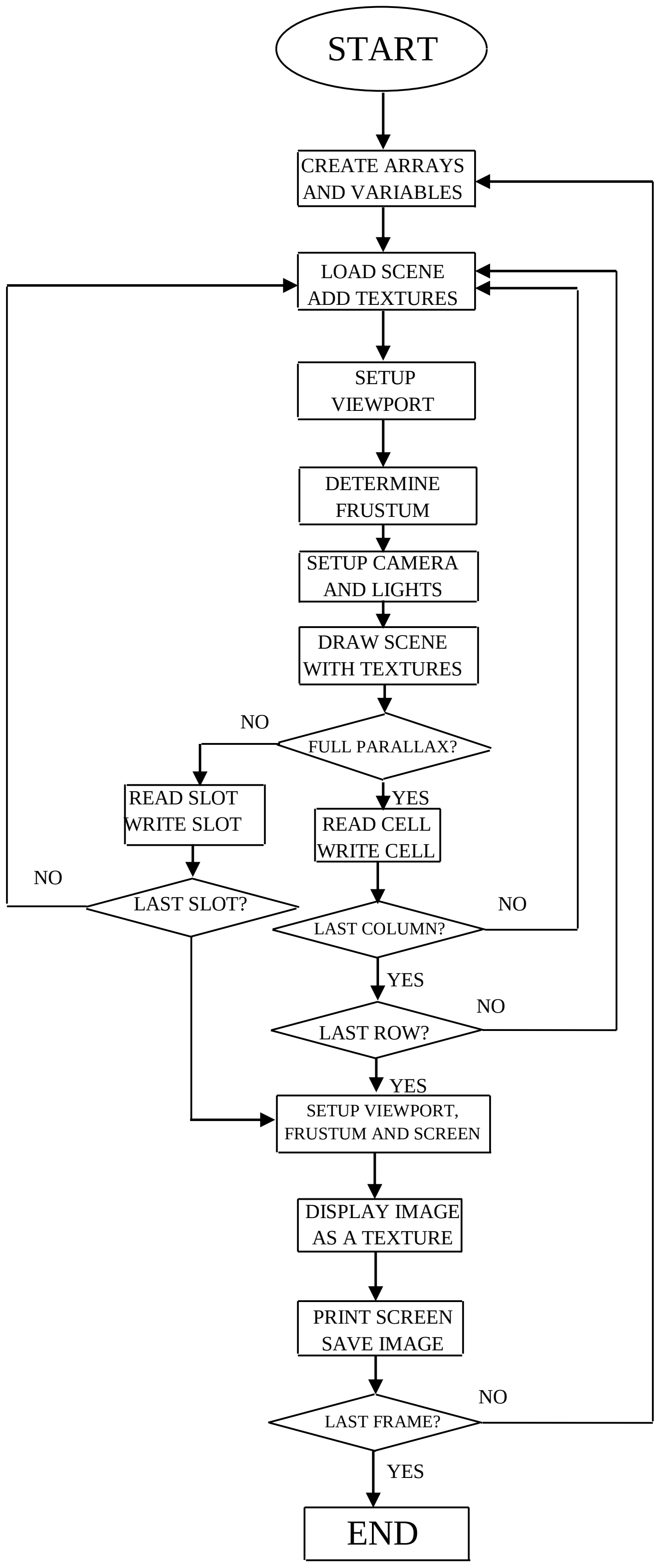


Figure 3.25: Computer generation of integral images flowchart. Based on *DIVGL* algorithm.

3.3.5. The implementation of the integral images computer generation method *DIVGL*:

Referring to Figure 3.25, at the beginning, the variables are initialised. These variables include the width and height of the display device screen on which the image is supposed to be displayed. The ID of the objects and textures are the pointers to the memory locations where the objects and textures data are saved. A dynamic array is available to be used to save the image, the size of this array should be equal to the number of pixels in the screen multiplied by the number of bytes employed to save each pixel. Normally, for an RGB pixel we need 3 bytes:
*unsigned char *pixels_index = new unsigned char[3×width×height];*

To add a texture we need to save an image in memory as a texture and return the ID of the texture, the function *loadTexture()* takes the file name of the image and returns a texture handle which is the ID of the texture. The function *initRendering()* initializes the rendering parameters such as the Depth-Test that makes sure that the objects hide from view partially or entirely the objects situated behind them.

load () is another function called in the initialization process of the parameters. *load ()* is a function from the class *MD2Model*. It loads the scene file saved in MD2 format and returns the ID of the loaded scene or a pointer to the memory where the scene data are saved.

In this particular application, the scene file format is MD2 and the loading tool is capable only to load the MD2 files. The loading tool of the MD2 files or the files with the extension *.md2* is composed of two simple files, a header file with the extension .h called *md2_format.h* and a C++ file with the extension *.cpp* called *md2_format.cpp*. In the header file, the external variables needed to use globally in our application are defined such as the number of frames *numFrames* and *frame_index* that indicates the frame numbers. In addition, in the header file all the structures, and functions needed to load the MD2 files are declared. The *.cpp* file *md2_format.cpp* includes *162 x-y-z* coordinates for *162* normal used in MD2 files. In *md2_format.cpp* file, all the functions needed to load MD2 files, textures, frames, and animations are defined.

The function *handleResize (int w1, int h1)* sets up the view ports and the frustums depending on the required parameters. The function *glViewport ((column_index×w1), (row_index×h1),* *w1, h1)* sets up the dimensions of the viewport. The parameters of this function are respectively the *x* and *y* coordinates per pixels of the lower left corner of the viewport, and the width and height of the viewport. *column_index* and *row_index* are integer variables to indicate the number of the columns and rows of the lenses respectively under which the elemental image is located. In the case when cylindrical lenses are used, the number of pixels in the vertical direction or the height in pixels of the lenses will be the same as the height of the screen or the total image if it is different to the length of the screen.

Frustums are setup depending on the column index and row index as it is declared in the capture stage. The coordinates of the horizontal clipping planes that determine the frustums are calculated in the same way of the vertical clipping planes. $zNear$ and *zFar* should be calculated to be used in the algorithm. If the number of projections is $m = 8$, the number of horizontal lenses is *128*, the number of vertical lenses is *128*, the normalized lens pitch is $p = 2/n = 2/128 = 0.015625$, then, $zNear = m\times c\times F$ where F is the value of the focal length of the lenses or pinholes. $zNear$ must be measured in the same length unit that is used in the scene which was created in the 3-D application at first place. For example, if the dimensions of the scene objects are measured in centimetres, the far and near planes z-coordinates will be measured in centimetres. In this case, the focal length should be multiplied by a constant *c* that converts the value of $zNear$ to be in centimetres. In the previous example, the focal length is chosen to be 0.35 cm, in this case $c = 1$ and $zNear = 2.8$. *zFar* is selected so that the whole scene is contained in the view volume, for example *zFar = 200*.

The following loop is implemented to specify the frustums at each time we produce an elemental image for full parallax content to be displayed using a microlenses array.

```
if (column_index < 4)
        {
                        if (row_index < 4){
                        glFrustum(-1.0, -1 + p*(4.5 + column_index), -1.0, -1 + p*(4.5 +
                        row_index), 2.7, 200);}
                        if (3 < row_index < 125){
                        glFrustum(-1.0, -1 + p*(4.5 + column_index), -1.0 + p*(row_index -
                        3.5), -1.0 + p*(4.5 + row_index), 2.7, 200);}
                        else {
                        glFrustum(-1.0, -1 + p*(4.5 + column_index), -1.0 + p*(row_index -
                        3.5), 1.0, 2.7, 200);}
        }
if (3 < column_index < 125)
```

```
        {
                        if (row_index < 4){
                        glFrustum(-1.0 + p*(column_index - 3.5), -1.0 + p*(4.5 +
                        column_index), -1.0, -1 + p*(4.5 + row_index), 2.7, 200);}
                        if (3 < row_index < 125){
                        glFrustum(-1.0 + p*(column_index - 3.5), -1.0 + p*(4.5 +
                        column_index), -1.0 + p*(row_index - 3.5), -1.0 + p*(4.5 +
                        row_index), 2.7, 200);}
                        else {
                        glFrustum(-1.0 + p*(column_index - 3.5), -1.0 + p*(4.5 +
                        column_index), -1.0 + p*(row_index - 3.5), 1.0, 2.7, 200);   }
        }
else
        {
                        if (row_index < 4){
                        glFrustum(-1.0 + p*(column_index - 3.5), 1.0, -1.0, -1 + p*(4.5 +
                        row_index), 2.7, 200);}
                        if (3 < row_index < 125){
                        glFrustum(-1.0 + p*(column_index - 3.5), 1.0, -1.0 + p*(row_index -
                        3.5), -1.0 + p*(4.5 + row_index), 2.7, 200);}
                        else {
                        glFrustum(-1.0 + p*(column_index - 3.5), 1.0, -1.0 + p*(row_index -
                        3.5), 1.0, 2.7, 200);}
        }
```

The function *drawScene()* performs several actions using OpenGL, GLUT, and GLU libraries. For example, clearing buffers and setting the current matrix mode are performed with the functions *glClear(),* and *glMatrixMode(GL_MODELVIEW)* respectively. The later one performs subsequent operations on the matrix *GL_MODELVIEW,* translate and rotate the camera to a specific position with *glTranslatef,* and *glRotatef* respectively. The functions *ambientLight*, *lightColor* and *lightPos* are called to set the ambient light, the colour of light and the position of light in the scene respectively. The function *draw()* is a member function of an object from the class *MD2Model* pointed to by a pointer defined at the beggining. *draw()* draws the objects after applying on them rotation, translation and scaling if needed. The textures that were loaded previously are mapped to the scene. The function *update (int value)* will be called to repeat the cycle of loading scene and setting up the parameters of the viewport, the frustum etc... The integer value is an ID of the function that is set to be *0*. If another similar function is needed, a different ID must be specified. Each time this function is called, the previous step is implemented and new variables are created. In the case when the rendering is meant to be for full parallax, an array is created to hold the pixel values of the elemental image with specific row and column index. *glReadPixels()* allows to read a specific area in the rendered image which is the area on the screen where the elemental image is displayed. In the case of full parallax, the pixels will be read from the screen pixel by pixel and mapped into the final image. The way of mapping is depending on the image we need to display weather it is a virtual orthogonal image or a real pseudoscopic image.

To display a virtual orthogonal image we need to rotate the pixels with 180º around the centre of the elemental image, whereas, we need to map the pixels as they were read from the screen without rotation if we want to display a real pseudoscopic image. The elemental image that is read from the screen is saved in the temporary dynamic array that is locally created each time we call the function *update()*, and then it is saved in the array that was created at the beginning. Pixel values with a size of 3 bytes each are saved in the array starting from the RGB values of the first pixel on the bottom left corner of the screen. We save pixels on the first row of the screen starting from the bottom left to the bottom right corner of the screen. The second row to scan is the second line above the bottom line starting from the left pixel and so on. For each micro image, we need to read the image pixels from the screen and map them into the final image. If *xi* and *yi* are the local coordinates of the pixels of the elemental image, *w_temp* and *h_temp* are respectively the width and height of the elemental image. Each pixel is represented with 3 bytes, so the local index of a pixel within the local array is:

$$local\_index = 3\times(yi\times w\_temp + xi) \quad (3.20)$$

The global index of the same pixel in the final image is:

$$global\_index = 3\times(row\_index\times w\times h\_temp + yi\times w + column\_index\times w\_temp + xi) \quad (3.21)$$

The pixels are mapped to the image, each colour value is written with a byte of the array. The first colour R (red) value, G (green) value, and B (blue) value are assigned to the pixeles indexed with *local_index*, local_index +1, and local_index +2 respectively as follows:

$$pixels\_index[global\_index] = pixels\_index\_temp[local\_index]; \quad (3.22)$$
$$pixels\_index[global\_index + 1] = pixels\_index\_temp[local\_index + 1]; \quad (3.23)$$
$$pixels\_index[global\_index + 2] = pixels\_index\_temp[local\_index + 2]; \quad (3.24)$$

The resulting image is displayed as a texture. A new viewport is specified to hold the whole image, and then the displayed image is printed and saved as a frame. If the scene is an animated

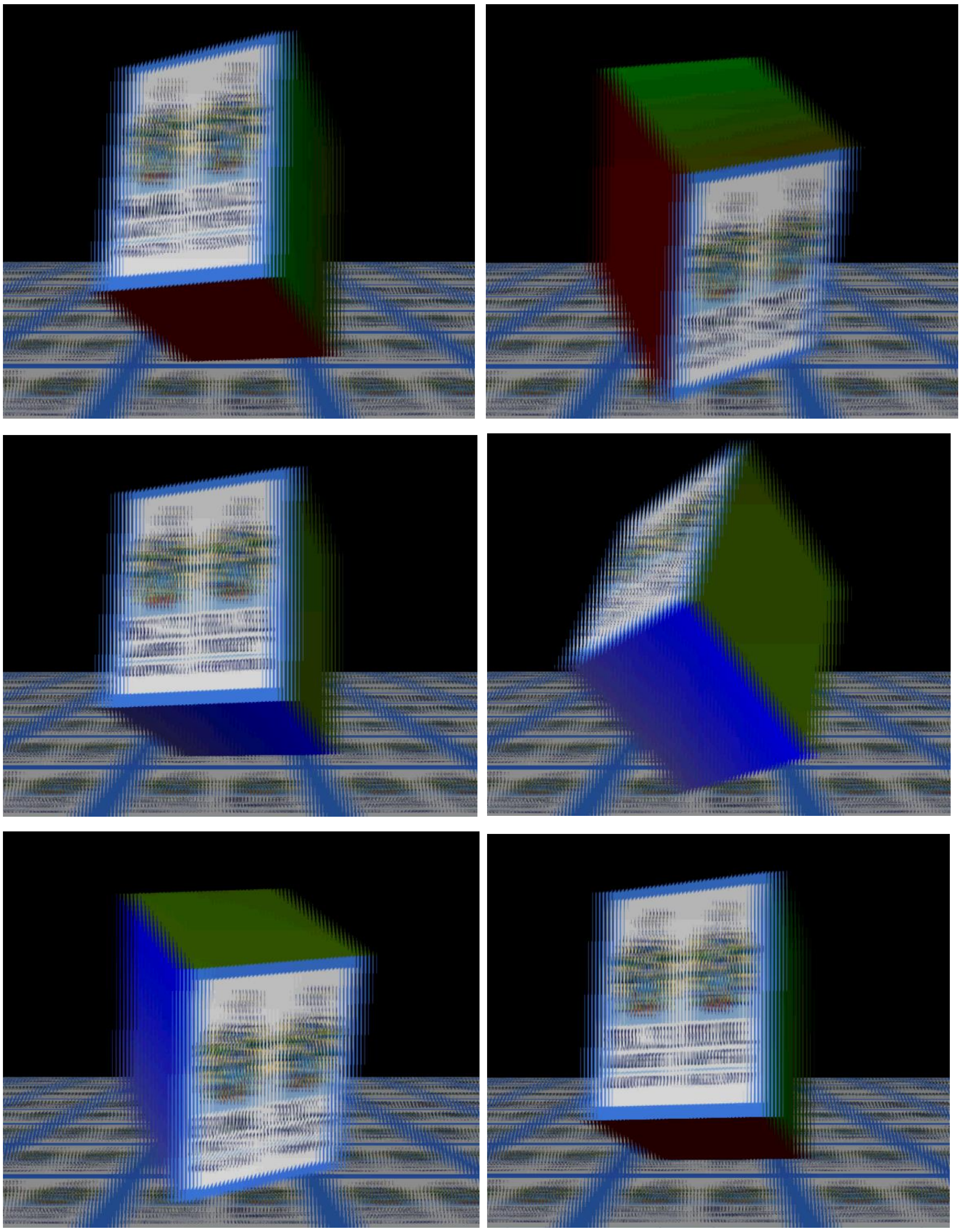


Figure 3.26: 3-D animated integral images generated using *DIVGL* algorithm.

3.3.7. The scene dimensions and the object points coordinates

scene, each frame of the scene will be displayed apart and saved as a separated image. The total collection of frames are imported to a video application such as Windows Live Movie Maker which converts the frames to a movie to be displayed on a screen covered with the suitable lens array. The image shown in Figure 3.21 is supposed to be displayed with microlens array. In the code used to render this image, *w_temp,* and *h_temp* are temporary width and height set to be *8* and *6* respectively. The small window of *8×6* pixels is scanning the image horizontally from the lower left corner to the right and then vertically form bottom towards the top until it reaches the upper right corner, therefore, the number of bytes in memory will start with *0* and ends up with *3×1024×768*. The OpenGL function read the pixels values of the small screen in the same way (i.e. lower left to upper right indicated with the local index), save them in an array of *3×8×6* elements, and then with the FOR loop the values are saved in their corresponding location in the main array *pixel_index*. After scanning the whole image the resulting array *pixel_ screen* that holds the pixels values will be displayed on the screen using OpenGL functions after the row index reaches *128* and get out the IF statement. The image will be loaded from an md2 file (e.g. *tris0.md2).*

3.3.6. Generating animated scenes with *DIVGL* algorithm

Animated scenes can be generated with *DIVGL* method by generating each frame of the scene as a statics image and display the frames in sequence. Figure 3.26 shows samples from the frames of an animated scene of a rotating cube.

In OpenGL, we can define the shapes of individual objects within a separate coordinate reference frame for each object called local coordinates; the objects are placed into appropriate locations within a scene reference frame called world coordinates [5]. In the cutlery scene example, the individual objects are transferred in proportion to the scene coordinates using the function *glTranslatef (x1, y1, z1)*. The scene coordinates are selected in proportion to the world coordinates. The scene is transferred using *glTranslatef(x, y, z)* function so that *x, y,* and *z* are the coordinates of the origin of the local coordinates while *x1,y1,* and *z1* are the coordinates of one object in proportion to the origin *x, y, z*. The function *glFrustum(-1.0, 1.0, -1.0, 1.0, 2.2, 200)* is used to determine the valid space for projection, the last two parameters are the distances of the near and far clipping windows in proportion to the origin of the world coordinates.

The scene in OpenGL was composed of few simple objects to demonstrate the method of producing a 3-D integral image from a 2-D image. The 2-D image rendered with OpenGL in the environment of C++ is displayed on the computer screen. The resulting image is meant to be displayed as a 3-D spatial image using the suitable lens array positioned in front of the 2-D displayed image. OpenGL coordinates are *x, y,* and *z*, the centre of the coordinates or the point where *x=y=z=0* is located at the centre of the screen. X-axis and Y-axis are the 2-D axis of the screen. X-axis is the horizontal one directed from left to right; therefore, the positive direction is from left to right. Y-axis is the vertical axis, the positive space is the upper one and the negative space is the lower one. Z-axis is the perpendicular axis on the screen directed to the positive space towards the viewer while the negative direction is the opposite. The camera is always located at the origin of the virtual view; the camera is simply the viewer. The camera has two coordinate systems, the local and the world coordinate system. The local coordinate system in which the camera positions at its origin point (*0, 0,* and *0*) can be used when the camera is rotated as it is rotates around its local coordinates. Whereas, the world coordinates are used when the camera moves and moves its world coordinates. In general, we have local and world coordinate systems. Objects are positioned in both the local and the world coordinate systems. The local coordinate system is used for rotating objects, whereas, the global coordinate system is used for positioning objects in the world [57].

Taking the scene of Figure 3.14 as an example, one of the objects in the scene is the (spoon) in Blender, the length of the spoon is *315* units; the highest width is about *60* units. The object is scaled in OpenGL with the factor of *0.05* for *x, y* and *z,* which makes the length and width *15.75*, and *3* respectively. This particular object in this example is located at the OpenGL coordinate *z = -10* after translation, while the OpenGL camera is located at local coordinate *z = 0*. We usually suppose that the screen is located at the same *z* coordinate of the camera, the camera in OpenGL and as a result, the scene in the computer screen can be transformed by changing the camera location, rotating around the local coordinates, translating the objects in the world coordinates etc... The scene is translated to *z=-20* using the function: *glTranslatef(0.0f, 0.0f, -20.0f)*. In this case, the whole scene is moved by *20* GL units to the negative space, therefore, the camera is at *z=20* in GL coordinates. If we suppose the origin of local and world coordinates are identical at the beginning, the camera coordinates will be moved to GL coordinate *z=+20*, the whole scene with be shifted by *20* negative GL units in the camera relative space.

Once the scene has been modelled in world coordinates, a viewing coordinate system is selected and the description of the scene is converted to viewing coordinates. The viewing coordinate system defines the viewing parameters including the position and the orientation of the projection plane (i.e. view plane) which we can think of as the camera film plane [5]. The view plane in our case is simply identical to the computer screen plane; the clipping window is the selected window in which the OpenGL 2-D image is displayed. The length and the width are specified with OpenGL window size in pixels, in this example the size is *1024 x 768* pixels. The view volume is the 3-D clipping region in space where the models are located. The size of this region is defined by the selected clipping window and determined by the near clipping plane and the far clipping plane, (see Figure 3.9).

An angle of $\theta = 45^o$ is selected to be the field-of-view angle from which the symmetric perspective projection frustum view volume is seen. The GLU function that sets up the view volume is *gluPerspective(45.0, 1024 / 768, 1, 200)*. The distance of the far clipping window and the near clipping window from PRP are respectively *200* and *1*, whereas, the proportion *1024/768* determines the dimensions of the clipping window in pixels.

The scenes used as examples are originally created in 3Ds Max or Blender. These scenes had to be exported from either Blender or 3Ds Max to the environment of Visual C++, processed, and displayed with OpenGL. The models are loaded to C++ using the MD2 files loader, whereas, the texture images of the models in the format .bmp are loaded using the image loader.

3.4. Summary

The Equations derived in other works that are employed to produce integral images in the way of forward projection can be modified, adopted, and employed to produce integral images with OpenGL in the environment of VC++. Both the finite-sized and the pinhole projection rendering models can be implemented using these Equations with some modifications. When implementing the finite-sized projection rendering model, errors occurred due to the shortage in the triangles or the vertices forming the mesh of the object and the complexity of the calculations. The integral image produced with the forward

projection pinhole rendering model is more accurate and more realistic. 3D-from-2D image was successfully produced by using the forward projection pinhole rendering model for each projection point of 8 projections and multiplexing the sub-images of the projections. Using OpenGL and VC++ to produce integral images in both the two models is much easier, faster, more flexible and more accurate than applying immediate calculations on the vertices of the projected object at a lower level. The functions and the methods used in the programs reduce the efforts, save the time needed, and produce better integral images at least with the pinhole mode or 3D-from-2D model. In addition to the main operations that are needed on the object mesh, other sub-processes were created in the scene at a low level (i.e. the vertices level) such as lighting, shading, etc... These are now implemented professionally at a higher level with better accuracy and quality using OpenGL functions.
The successful production of a static integral image with the multi-projection pinhole model can be generalized to produce animated integral images with the same way. The main issue with producing animated integral images with this way is the time. To produce a 3-D animation with this method, a faster and more powerful computer must be employed. Producing video images with this approach simulates the process of filming real animated scenes with a 3-D video camera that is feasible according to the results of the experiments explained above. With the method of integral images computer generation (*DIVGL*), autostereoscopic images and films with 3-D effect can be produced by creating the scenes and animations in 3-D applications without the need for physical equipment neither in the capture stage nor in the depth inverting stage. The primary image resulting from the capture stage is pseudoscopic but it does not need to be corrected for converting it from pseudoscopic to orthographic using physical devices. The image can be simply corrected by mapping the pixels in the image plane to produce virtual orthogonal images of the scene. Simulating the pickup stage allows the images to avoid the possible degradation can be caused by the different stages of capturing and correcting the image.

The method *DIVGL* produces integral images of scenes composed of several objects despite of the limitation in resolution the record and display devices can provide. If the whole scene is recorded, a very high resolution is needed to represent the details of the scene, whereas, the resolution of normal devices is limited. Resolution is primarily limited by the number of pixels for each elemental image in the pickup devices as well as the display devices. The resolution of the available printing devices is much more than that of the record or display devices, therefore, this method is suitable for producing printed integral images.

Chapter 4

Computer Generation and Rendering of Integral Images by Displacing the Virtual Camera Target (DCTarget)

4.1 Introduction

Images with *3-D* effect are more realistic and closer to the real world images as they create an influence on the human eyes similar to that created by real objects. In general, *3-D* images and videos are preferred over the traditional *2-D* images. There is growing evidence that *3-D* imaging techniques will have the potential to establish a future mass-market in the fields of entertainment (TV, video game) and communications (desktop video conferencing) [1].

Adding a third dimension to the traditional *2-D* images and videos was the subject of many different approaches and inventions. In several types of *3-D* images and videos, viewers need to wear special glasses to be able to see the *3-D* effect. The need for such a device can form a practical obstacle in the situations when the viewer is not prepared to watch *3-D* images. For example, if *3-D* images are meant to be used for advertisement. In order to implement free viewing *3-D* display, different methods were introduced aiming to produce auto-stereoscopic images. Several groups have demonstrated auto-stereoscopic *3-D* displays [2]. True auto-stereoscopic *3-D* display systems should have parallax in all directions and present images [1]. Multi-view Imaging (*MI*) and holography are popular methods for realizing auto-stereoscopic images. Auto-stereoscopic display based on the *MI* principle does not produce a fully accurate *3-D* representation of the scene [3].

3-D Holoscopic imaging, also referred to as Integral Imaging (*II*) is a technique that is capable of creating and encoding a true volume spatial optical model of the object scene in the form of a planar intensity distribution by using unique optical components [10]. *II* provides auto-stereoscopic intensity images with full parallax, free of any viewing device [4]. In *II* technique, a micro lens array is used to capture the image so that each lens views the scene from a different point and parallax information is recorded. Images with a *3-D* effect can be viewed when the parallax information is replayed. *II* has a number of advantages over both holography and *MI*.

In this chapter, the method of Computer Generation and Rendering of Integral Images by Displacing the Virtual Camera Target (*DCTarget*) is proposed. *DCTarget* method is aimed to generate static and animated *3-D* auto-stereoscopic integral images. The resulting *3-D* images and videos can be viewed with naked eyes without the need for special glasses. These 3-D images can be displayed on a special screen provided with a spherical micro lens array or lenticular lens array (i.e. cylindrical lens array).

The integral image is composed with pixels selected from a number of orthographic images. Each orthographic image is the orthographic projection of the scene on an image plane rotated with a specific angle that is equal to the rotation angle of the virtual camera target. A plug-in tool or so called Application Software (*APS*) based on the methods *DCTarget* and *DIVGL* is designed to convert computer generated *2-D* images and animations to static and animated *3-D* integral images. *2-D* images are the input of the *APS*. The parameters of the virtual system are selected through Human Machine Interface (HMI) that is available in the *APS*. The algorithm *DCTarget* or *DIVGL* is implemented and applied to the input images, and then the required *3-D* integral images are generated. The *APS* allows the user to tune the rendering and system variables through the HMI. These variables include the lens pitch, lens focal length, the number of lenses, the type of the virtual lens array used in the capture stage as well as the display stage, the type of rendering, and other parameters. *DCTarget* algorithm that is implemented with the *APS* can provide fast generation of high quality integral images.

Thomas and Stevens [3] used multiplexed orthographic projections to display a *3-D* model as a *3-D* image, the images. However, images produced with this method can be classified as a *MI* technique, and therefore, the *MI* disadvantages are available, whereas, in the *DCTarget* method, pixels are selected from several orthographic images and mapped to a single image. *DCTarget* is an *II* technique and the advantages over *MI* are available. R. Olsson and Y. Xu [13] created a simulation environment to allow for a simple definition of complex scenes and from those, integral images are synthesized. *DCTarget* with the *APS* and the graphical user interface (*GUI*) is simpler and more flexible as a wider range of *2-D* static and animated scenes can be converted immediately to images and videos with *3-D* perception. Sokolov [15] proved that a pinhole array could be used to capture a scene and create integral images. This model was implemented with a software model and integral images were produced and displayed on the flat surface of a micro lens array [51]. *II* is one of the *3-D* imaging technologies based on a pinhole array or a cylindrical lens array to capture incoherent light rays from different directions [31]. Pinhole array is considered and simulated in *DCTarget* method as the tool used in the image capturing stage of the process that is aiming to produce integral images. *DCTarget* is a fast, cheap and efficient way to produce high quality integral images with the ability to select the parameters of the imaging system. *DCTarget* method comprises two stages, the capture stage and the replay stage. The capture stage is simulated with the application software, the output of this stage is the computer generated integral image. The replay stage is implemented by displaying the computer generated integral image on a screen provided with optical tools; the integral image is reconstructed and displayed as a 3-D auto-stereoscopic image.

4.2 The pickup stage

We need to prove that the micro-image formed under each pinhole when a pinhole array is used to capture the scene (or microlens when a microlens array is used) is a perspective image of a part of the scene, and then, the micro-image composed from pixels selected from different orthographic projections of the scene are perspective images. In other words, these micro-images are the same as the micro-images produced with *DIVGL* method.

Each lenslet produces a micro-image provided with the characteristics of a perspective micro-image. If a micro-image formed under each lenslet is provided with X' and Y' axis, x', and y' coordinates of the object point projection are inversely proportional to z coordinate of the projected points in the scene. The expression for the x', and y' display coordinates stated in OpenGL for the perspective camera transformation suggest the same relationships between x & z, and y & z in the horizontal and vertical directions respectively [54].

If the camera is located on the focal point of the lens or the pinhole that is considered as the centre of the coordinate system, the coordinates of the object points and their projections on the image plane are linked with the following formulas:

$$x' = ((-a \times x)/(b \times z)) + c \qquad (4.1)$$

$$y' = ((-a' \times y)/(b' \times z)) + c' \quad (4.2)$$

If each micro-lens is considered as a separated camera, for each micro-lens, the coordinates x′ and y′ of the projection points on the image plane are shifted with x1 and y1 respectively where x1 and y1 are the coordinates of the micro-lens in the image plane.

In the image plane, the centre point in the image plane is the centre of the coordinate system. The coordinate system with X′ and Y′ axis is a local one for each micro-lens in which the centre of the micro-lens is the centre of its local coordinate system. The previous relationships are correct for the local coordinates of each micro-lens. To make these relationships correct in the global coordinate system of the image plane, the coordinates of the micro-lens should be considered. The coordinates of the object points would be shifted with x1 and y1, and then the relationships between the coordinates of the projection points in the image plane and object points are:

$$x' = ((-a \times (x - x1))/(b \times z)) + c \quad (4.3)$$

$$y' = ((-a' \times (y - y1))/(b' \times z)) + c' \quad (4.4)$$

x′, and *y′* are the coordinates on the image plane, *x*, and *y* the coordinates in the scene, *a′*, *b′*, *c′*, *a*, *b*, and *c*, are constants related to the image settings.

In our case, each lens is considered as an independent camera, then, each lens has an independent coordinate system with a centre located on the focal length of the lens, whereas, the centre of the coordinate system in which the coordinates of the object points are measured is located on the centre of the image plane (i.e. point *c*). Therefore, to use the same coordinate system for the object points and their projections, the object points should be transferred to the coordinate system of each lens, and the Equations 4.3 and 4.4 will be as follows:

$$x' = -a \times (x - c1)/b \times z + c = ((- x + A)/(B \times z)) + C \quad (4.5)$$

$$y' = -a' \times (y - c2)/b' \times z + c' = ((- y + A')/(B' \times z)) + C' \quad (4.6)$$

where, *A, B, C, A′, B′,* and *C′* are constants related to the image settings. If a cylindrical lens array is used, (4.5) is applicable, whereas, the point with $y = 0$ in the central vertical line of the lenslet is considered as the centre of the lenslet local coordinates. The centre of the local

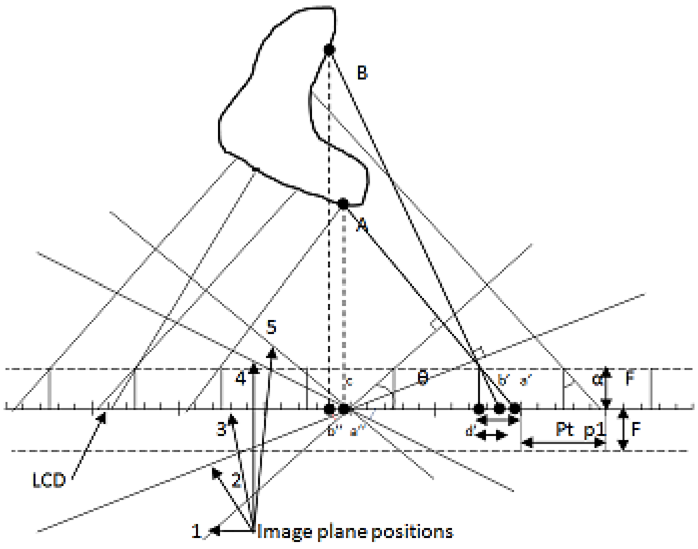


Figure 4.1: The micro-image is a perspective projection of a part of the scene.

coordinate system is shifted in x direction only and then Equation (4.6) would be:

$$y' = (- y/(B' \times z)) + C' \qquad (4.7)$$

Referring to Figure 4.1:

$$a'd'/a'a'' = F/Aa'' \qquad (4.8)$$

$$a'd' = a'a'' \times F/Aa'' \qquad (4.9)$$

$$x' - cd' = (x'-x) \times F/z + F,\ cd' = c1 \qquad (4.10)$$

$$x' \times z - c1 \times z + x' \times F - c1 \times F = x' \times F - x \times F \qquad (4.11)$$

$$x' \times z - c1 \times z = -x \times F + c1 \times F \qquad (4.12)$$

$$x' = ((-x \times F + c1 \times F)/(z)) + c1 \quad (4.13)$$

$$x' = ((-x + c1)/(1/F \times z)) + c1 \quad (4.14)$$

The same discussion is valid for y′, if the constants are replaced, the last Equations for x′ and y′ will be as follows:

$$x' = ((-x + A)/(B \times z)) + C \quad (4.15)$$

$$y' = ((-y + A')/(B' \times z)) + C' \quad (4.16)$$

In the case of using a cylindrical lens array, (4.6) can be written as $y' = -y/B' \times z + C'$, where, x′ and y′ are the projection coordinates of the object points with the coordinates x and y in the same coordinate system. *A, B, C, A′, B′,* and *C′* are constants. Equations (4.5) and (4.6) are identical to Equations (4.15) and (4.16) respectively, for each micro lens *A*, and *A′* mean that x coordinates of the object points are shifted but that doesn't affect the relative positions of the object point projections. Therefore, the micro-image formed under each pinhole is a perspective image of a part of the scene. The capture stage is a simulation of the process of creating a cluster of micro-images of the scene, which is processed and used to generate integral images. In order to create integral images with horizontal parallax, cylindrical or Plano-convex cylindrical lens array can be used. If the lens array is placed on the image plane, which is an array of pixels charge-coupled device (*CCD*) employed to record the cluster of micro-images formed by the lens array, each lenslet of the array captures a part of the scene and creates a micro-image to be recorded on the image plane. Full parallax can be achieved using a spherical microlens array. A simplification of the process is applied by replacing the microlens array with an array of pinholes; also, the simulation of the imaging process is based on this simplification. Each microlens (or pinhole) plays the role of a mini camera, and each micro-image is a perspective projection of the scene or part of the scene that can be seen through the aperture of that mini camera.

Referring to Figure 3.8 in which the pickup stage and the replay stage of the integral imaging process are shown. Each pinhole (or micro lens) in the pickup stage sees a part of the scene called the micro lens field of view. The micro-image formed by each lens is the image of that part of the scene the lens sees, which is the entire part of scene that exists in the lens's field of view. It is possible to consider the micro-image formed by each micro lens as a perspective projection of the points of the objects in the scene on the image plane through the lens or the aperture of the pinhole providing that the points of the objects exist in the lens's field of view. Regarding the fact that the surface of the image plane on which the micro-image is recorded contains a limited number of pixels, it is possible to trace back all the rays of light received by the image plane starting from the pixels of the micro-image and ending with the projected scene points. The incident rays striking the image plane at the extreme left and right pixels of the micro images are shown in Figure 3.8.

In order to simplify the calculations in the pickup stage, the micro lens array is replaced by a pinhole array. Using a pinhole array is an embodiment of the method, however, using microlens array is supposed to produce images with higher quality. In the case of using microlens array, the refraction of the rays would be taken into account when the pixel values are acquired and calculated. As a trade-off between the quality and the calculation complexity, the pinhole array is considered instead of microlens array and the image pixel values resulting in this approach are approximated image pixel values supposed to be acquired when using microlens array. One of the tasks the application software can perform is the process of generating the integral image by creating a cluster of micro-images in which the number of micro-images is equal to the number of spherical or cylindrical micro lenses used to capture and replay the integral image. One perspective projection is needed to create each micro-image. If the number of micro lenses is 100×100 = 10000, the number of projections needed is 10000. In order to reduce the number of projections and perform faster and easier process of rendering the integral images, the method of displacing the virtual camera target is introduced.

The camera target is a point in the scene space at which the camera is focusing and the camera central axis is the vertical axis that is perpendicular to both the lens array flat surface and the image plane. The intersection point between the flat surface and the camera central axis is the centre point. In this method the virtual lens array (or the virtual camera) that is utilized to capture the micro-images is rotated so that the camera target is displaced, whereas the centre point is fixed. The global coordinates of the centre point are (*0, 0,* and *0*). The rotation angle of the camera central axis is equal to the rotation angle of the image plane and the lens array. The initial position of the lens array and image plane is the position when the camera target has zeroed x, and y coordinates, the camera central axis has zeroed vertical and horizontal rotation angles, and the image formed on the image plane in this position is replaced with the Final Integral Image (*FII*). If the microlens array is used, the lens array and the camera central axis are rotated vertically and horizontally, and the camera target is displaced to occupy different positions within the x-y plane, whereas, the rotation angles are only horizontal when using lenticular lens array, and the camera target in this case is moving along the x-axis. The scene is projected orthographically on the image plane, and the resulting image is the orthogonal projection of the scene objects points on the image plane; the incident rays starting from the objects points and hitting the pixels on the image plane are parallel to the camera central axis and perpendicular to the image plane. Each time the micro lens array is rotated, the coordinates of the camera target are changed and the camera central axis rotation angles are revaluated, and a new orthographic image is rendered. The technique used in this method is a process to compose a perspective integral image that is called *FII* from the different orthographic images. Specific pixels of every orthographic image are selected and mapped by value to the *FII*. The process of mapping specific pixels from the orthographic projections to the *FII* simulates the perspective projection of the scene with the same lens array. The projections of the scene objects points on the *FII* that result from the mapping process are equivalent to the perspective projections of the scene objects points on the initial image plane with the same microlens array.

Figure 4.2 shows an upper view of a horizontal cross section of the image plane and the imaged volume containing the objects. The camera's coordinates are permanently fixed while the direction of the camera is changeable. If OpenGL is the application programming interface used to display the images, the camera is always located at the eye space coordinate (0, 0, 0) which is supposed to be the centre of the display screen, or more precisely, on the centre of the image. Therefore, in order to view the rotation of the camera in the opposite direction, we need to rotate the scene while the camera stays in the same place, also, in order to view the translation as if the camera is translated in the opposite direction, the scene must be translated instead, while the camera is fixed. However, it is possible to consider the camera to be rotating or moving while the virtual scene is fixed, whereas, the scene is actually rotating and moving, while the camera location is in stationary. That is acceptable because moving /rotating the scene and moving/rotating the camera are identical in the viewer point of view.

Plane *xi*-*yi* of the image coordinate system contains the plane of the display screen. The horizontal axis is *xi*, the vertical axis is *yi*, and *zi* is the axis that is perpendicular to the image plane crossing it in the centre (*0, 0, 0*). The camera is located in that centre, the *zi* axis is directed to the opposite direction headed away from the viewer, therefore, *z* values of the points inside the screen are positive and the ones outside the screen are negative. In general, the following four coordinate systems are considered: 3-D world coordinate system (*x, y, x*), 3-D camera coordinate system (*xc, yc, zc*), 2-D image coordinate system, and 2-D pixel coordinate system. In this method, only the orthographic projection is considered. The image plane is supposed to be located in the *xi*-*yi* plane, and contained in the display screen or forming a part of it.

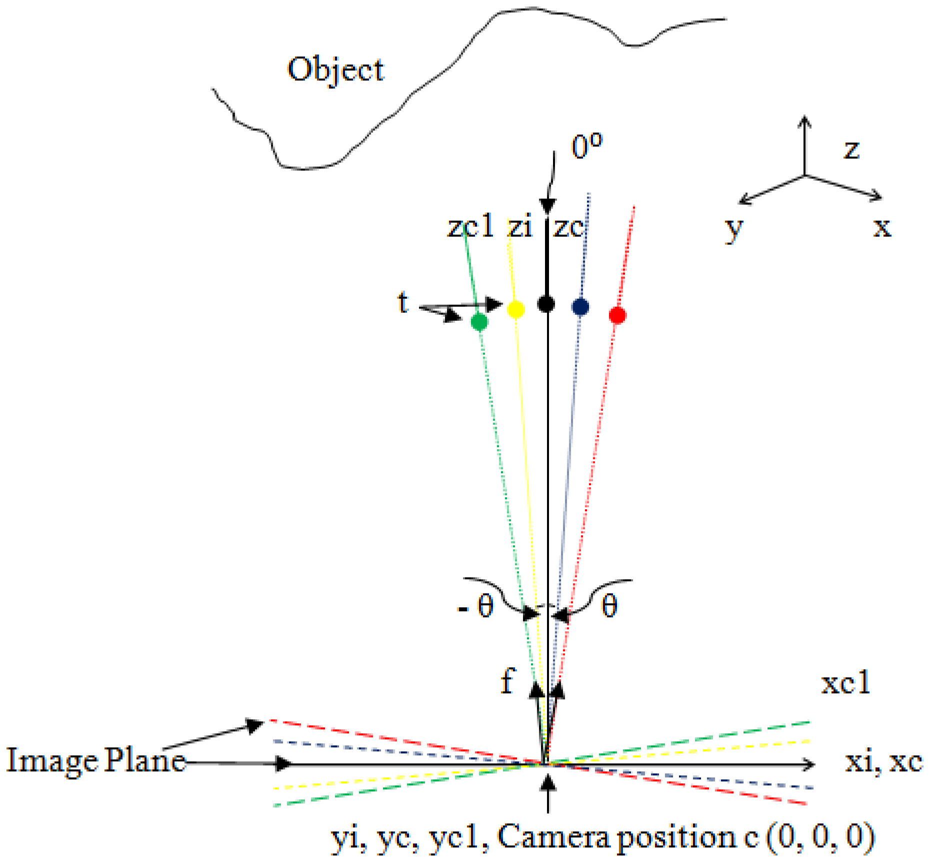


Figure 4.2: A horizontal cross section showing a camera with different image planes.

UV plane of the 2-D pixel coordinate system is contained in the screen plane. The origin of this system is the upper left corner of the image, U axis is directed from left to right, and V axis is directed from up to down. The measurement unit in this system is pixel, therefore, each point in the image has u and v coordinates measured by pixels and always positive. Figure 4.2 shows an object that simulates a real life object, but in this method, the objects are virtual and created in a graphic application such as 3-D Max, Maya, or Blender, and exported to VC++ as geometry and texture files. The vertices of the objects in the scene are supposed to have coordinates in the world coordinate system (x, y, and z), while the texture files contain u-v mapping of the image.

The scene and all the objects inside the viewing volume are orthographically projected on the image plane. The object coordinates are multiplied by the Model-View matrix to transform the vectors of the object from the object space where the world coordinate system or the object coordinates is used to the eye space or the camera space where the camera coordinate system is used. In addition, other elements in the scene such as lighting are transformed from the object space to the camera space. The Model-View matrix is the model matrix multiplied by the view matrix; the model matrix is to convert from object space to world space, while the view matrix is to convert from world space to camera space. Three elements are able to form the Model-View matrix; these are the position of the camera or the viewing eye, the look-at point to which the camera is pointing, and the UP direction vector. These elements can be used to construct the model-view matrix without using OpenGL transform functions.

The vector t-c from the camera point c to the target point t in each position of the image plane in Figure 4.2 is the look-at vector that defines the target point in proportion to the camera or the direction of the camera, to get the forward vector f with a unit length, the vector t–c is normalised. The vector (f) with its X, Y, and Z coordinates respectively (*f0, f1,* and *f2*) is always perpendicular to the image plane or the plane x-y of the camera coordinate system. *UP* vector with its X, Y, and Z coordinates respectively (*up0, up1,* and *up2*) is given to define the upper direction of the scene. It is not necessary for *UP* vector to be perpendicular to the forward vector (*f*), however, the default coordinates of this vector in the world coordinates are (*0, 1,* and *0*). In order to get the actual up vector, a cross product is applied to vector (*f*) and the upper vector (*up*). The resulting vector is the side vector (*s*) with its X, Y, and Z coordinates respectively (*s0, s1,* and *s2*) which is perpendicular to (*f*) and (*up*) vectors, and then a cross product is applied to side vector (*s*) and (*f*). The actual up vector is perpendicular to f and side vectors at the same time; therefore, the actual up vector is contained in the image plane. The three vectors: (*f*), side, and the actual up vector form the camera coordinate system in which the camera is located in the centre of this system.

The image plane is always orthogonal to the forward vector. The target point can be any point in the object space and therefore the image plane can be any plane that contains the camera location c. The camera in this method is orthographical one, which means the scene is projected orthogonally on the image plane.

Each point in the world space is multiplied by the Model-View matrix, in other words, each point is brought to the camera space. If the homogenous coordinate w equals to 1 and the most right column which is specified for the translation transformation is zeroed. The Model-View matrix [57] can be applied on written as follows:

$$\begin{vmatrix} s0 & up0 & f0 & 0 \\ s1 & up1 & f1 & 0 \\ s2 & up2 & f2 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Figure 4.3 represents these vectors, the camera space and the positions of the image plane.

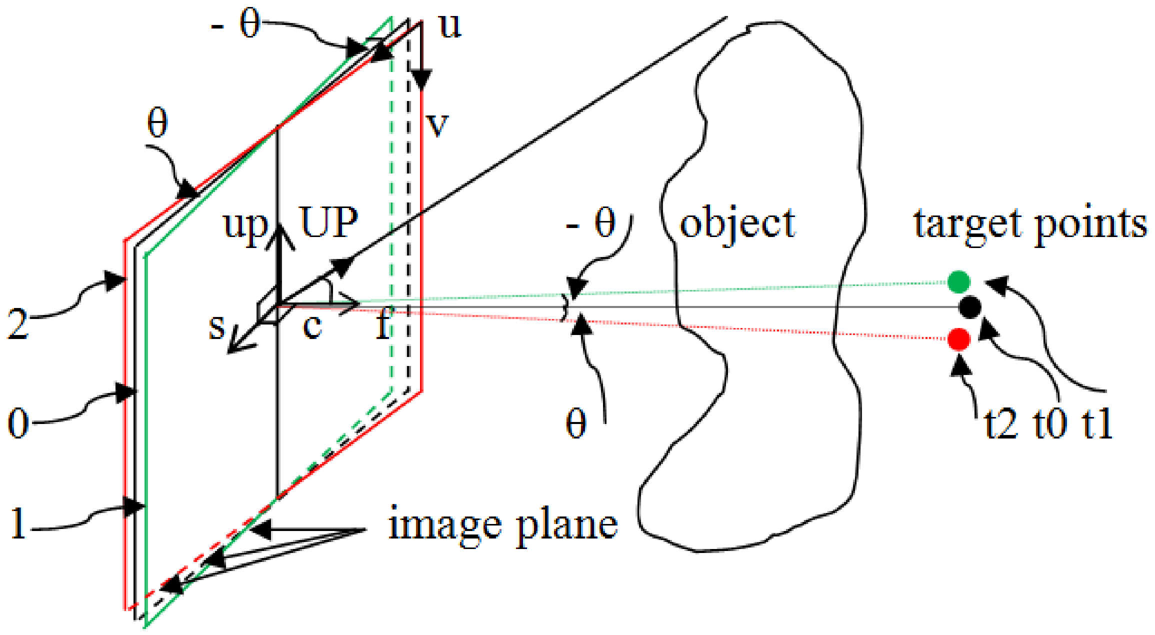


Figure 4.3: The camera space and *3* virtual positions of the image plane.

After multiplying the object vertices by the model-view matrix, the resulting coordinates are multiplied by the projection matrix to get the clip coordinates. In this step, object vertices are clipped out from the viewing volume or the frustum. For orthographic projection or the parallel projection, the clipping volume is a parallelepiped. The clipping volume has 6 clipping planes: front, back, left, right, near and far. In this method, the camera is positioned on the origin that is the centre of the near clipping plane or the front-clipping plane, the front plane is contained in the image plane.

Referring to Figure 4.3, the camera is located in the image plane and the front clipping plane, if the target point of the camera or the look-at point rotates with an angle ($-\theta$) keeping the camera in the same location, the forward vector (*f*) stays perpendicular to the image plane or the front clipping plane. In fact, to move or rotate the camera, the camera stays in the location with the coordinates (0, 0, and 0) while the scene moves, or rotates in the opposite direction. Because the movement or the rotation of the scene will be equivalent to the movement or rotation of the camera with an opposite direction, in order to make the discussion easier, from now on, virtual camera movement and rotation will be considered instead of moving and rotating the scene. Therefore, if OpenGL is used, moving or rotating the camera and image plane is equivalent to moving or rotating the scene respectively with the same amount but in the opposite direction.

In Figure 4.3, if the normal to the image plane numbered with *0* is supposed to be the reference normal, the angle between this normal and the forward vector *f* is equal to *0* when the virtual image plane is on the position *0*, and the target point is *t0*. If the camera stays in the origin and the target point moves to the point *t1* on the same horizontal plane formed with the vectors *f* and *s*. The angle between the reference normal and the forward vector *f* is $-\theta$. The virtual image plane will be rotated around *up* vector by the angle $-\theta$ to the position *1*, this direction is considered as the negative direction. If the target point moves to the position *t2* on the same horizontal plane, the angle between the reference normal and the vector *f* is θ, the virtual image plane is rotated around *up* vector to the position *2*, this direction is considered as the positive direction. The image plane always is formed with the vectors *up* and *s*, while vector *f* is always perpendicular to *up* and *s* vectors. The orthogonal projection is formed by projecting the vertices of the objects that are contained in the view volume on the image plane. Projection rays are parallel to the normal and perpendicular to the image plane.

4.3 The new target point calculation in relation to the rotation angles of the camera

If the camera is rotated by the angle α around *up* vector, and the angle β around *s* vector, so that the rotation is horizontal, the new target point will be located in the plane that contains the vectors *f* and *s*. In this case, the new target point will be belonging to the plane that is perpendicular to the forward vector *f* and containing the original target point *t0*. In general, all the target points will be belonging to the crossing line between these two perpendicular planes. In the case of vertical rotation, the rotation will be performed around the vector *s* and the new target will be located on the plane that is perpendicular to the side vector *s* or perpendicular to the plane that contains the vectors *f* and *s*. This plane will be containing the original target point *t0*, the camera location and the forward vector *f*. All the target points in this case will be belonging to the crossing line between this plane and the plane that is perpendicular to *f* vector and containing the original target point *t0*.

In this method, for a given original target point *t0*, camera location, and rotation angle, the coordinates *t_newx*, *t_newy*, and *t_newz* of the new target point in the two cases of horizontal and vertical rotations of the image plane (or the camera) around the camera location will be calculated.

When the rotation is horizontal only, vertical only, and mixture of the horizontal and vertical, the new target point vector *t_new(t_newx, t_newy, t_newz)* will be extracted from the following Equations respectively:

$$t\_new = c + R\times f + R\times \tan\alpha \times s \tag{4.17}$$

$$t\_new = c + R\times f + R\times \tan\beta \times up \tag{4.18}$$

$$t\_new = c + R\times f + R\times (\tan\alpha \times s + \tan\beta \times up) \tag{4.19}$$

where, the vector *c* (*cx*, *cy*, *cz*) is the camera location in the world coordinate system.
The angle by which the camera (or the image plane) is rotated around the vector *up* or the horizontal rotation angle is α.

The angle by which the camera (or the image plane) is rotated around the vector s or the vertical rotation angle is β.

The forward unity vector *f* (*fx, fy, fz*) is the normalised vector of a vector starting from the camera and ending in the original target point *t0*.

The vector *s* (*sx, sy, sz*) is the side unity vector that starts from the camera location.

The vector *up* (*upx, upy, upz*) is the unity vector that starts from the camera location.

The distance between the camera *c* and the original target point *t0 (t0x, t0y, t0z)* is *R*. The value of *R* will stay the same as it was at the beginning when the target point was at the given original position. The value of R can be calculated with the following Equation:

$$R = \sqrt{(t0x - cx)^2 + (t0y - cy)^2 + (t0z - cz)^2} \quad (4.20)$$

The vector *ct0* between *c* and *t0* is expressed with the following x, y, and z Equation:

$$ct0 = (t0x - cx) \times x + (t0y - cy) \times y + (t0z - cz) \times z \quad (4.21)$$

The forward unity vector *f* in the world coordinate system (*x, y,* and *z*) is result of normalising the vector *ct0*, from Equation (4.21):

$$f = \frac{(t0x - cx)}{R} \times x + \frac{(t0y - cy)}{R} \times y + \frac{(t0z - cz)}{R} \times z \quad (4.22)$$

The vector *UP* is given to define the upper direction of the scene; the default coordinates of this vector in the world coordinates are (*0, 1,* and *0*). The side vector *s* is the cross product of *UP* vector and *f.* The coordinates of the side vector *s* can be calculated by multiplying *UP* vector with *f.*

$$s = f \times UP = (fy \times UPz - fz \times UPy) \times x + (fz \times UPx - fx \times UPz) \times y + (fx \times UPy - fy \times UPx) \times z \quad (4.23)$$

$UPx = 0, UPy = 1, UPz = 0,$ then the side vector is:

$$s = (-fz \times UPy) \times x + (fx \times UPy) \times z = (-fz) \times x + (fx) \times z = [-\frac{(t0z - cz)}{R}] \times x + [\frac{(t0x - cx)}{R}] \times z \quad (4.24)$$

The vector *up* is the cross product of *f* and *s*.

$$up = s\times f = (sy\times fz - sz\times fy)\times x + (sz\times fx - sx\times fz)\times y + (sx\times fy - sy\times fx)\times z \quad (4.25)$$

From Equations (4.24) and (4.22):

$$up = (- sz\times fy)\times x + (sz\times fx - sx\times fz)\times y + (sx\times fy)\times z \quad (4.26)$$

$$up = (-fx\times fy)\times x + [(fx)^2 + (fz)^2]\times y + (- fz\times fy)\times z \quad (4.27)$$

$$up = \{-\left[\frac{(t0x-cx)}{R}\right]\times\left[\frac{(t0y-cy)}{R}\right]\}\times x + \{\left[\frac{(t0x-cx)}{R}\right]^2+\left[\frac{(t0z-cz)}{R}\right]^2\}\times y + \{-\left[\frac{(t0z-cz)}{R}\right]\times\left[\frac{(t0y-cy)}{R}\right]\}\times z \quad (4.28)$$

$$up = [-(t0x-cx)(t0y-cy)/R^2]\times x + \{[(t0x-cx)^2+(t0z-cz)^2]/R^2\}\times y + [-(t0z-cz)(t0y-cy)/R^2]\times z \quad (4.29)$$

The scalar components on the axis x, y, and z of the vector *ct0* that starts from the camera position and ends in the original target point are $(t0x-cx), (t0y-cy)$, and$(t0z-cz)$. These components will be replaced by the following variables respectively $R_x, R_y, and\ R_z$ where:

$$(t0x-cx) = R_x, (t0y-cy) = R_y \text{ and } (t0z-cz) = R_z \quad (4.30)$$

Equations (4.20), (4.22), (4.24), and (4.29) can be rewritten as the following Equations (4.30), (4.32), (4.33), and (4.34) respectively:

$$R = \sqrt{(R_x)^2+(R_y)^2+(R_z)^2} \quad (4.31)$$

$$f = \frac{R_x}{R}\times x + \frac{R_y}{R}\times y + \frac{R_z}{R}\times z \quad (4.32)$$

$$s = -\frac{R_z}{R}\times x + 0\times y + \frac{R_x}{R}\times z \quad (4.33)$$

up = [- $R_x \times R_y/R^2$]×x +[$(R_x{}^2 + R_z{}^2)/R^2$]×y + [- $R_z \times R_y/R^2$]×z (4.34)

The coordinates in the world coordinate system of the new target point for given camera location, original target point, horizontal rotation angle, and vertical rotation angle can be derived from Equations (4.19), (4.30), (4.32), (4.33), and (4.34):

$$t\_new = cx\times x + cy\times y + cz\times z + R\times(\frac{R_x}{R}\times x + \frac{R_y}{R}\times y + \frac{R_z}{R}\times z) + R\times\{\tan\alpha\times(-\frac{R_z}{R}\times x + 0\times y + \frac{R_x}{R}\times z) + \tan\beta\times\{[-R_x \times R_y/R^2]\times x + [(R_x{}^2 + R_z{}^2)/R^2]\times y + [-R_z \times R_y/R^2]\times z\}\} \quad (4.35)$$

$$t\_new = cx\times x + cy\times y + cz\times z + R_x\times x + R_y\times y + R_z\times z - R_z\times\tan\alpha\times x + R_x\times\tan\alpha\times z - (R_x \times R_y \times \tan\beta/R)\times x + ((R_x{}^2 + R_z{}^2)\times\tan\beta/R)\times y - (R_z \times R_y \times \tan\beta/R)\times z \quad (4.36)$$

$$t\_newx\times x + t\_newy\times y + t\_newz\times z = [cx + R_x - R_z\times\tan\alpha - (R_x \times R_y \times \tan\beta/R)]\times x + [cy + R_y + ((R_x{}^2 + R_z{}^2)\times\tan\beta/R)]\times y + [cz + R_z + R_x\times\tan\alpha - (R_z \times R_y \times \tan\beta/R)]\times z \quad (4.37)$$

Then the new target coordinates can be extracted From Equations (4.30) and (4.37) as follows:

$$t\_newx = t0x - R_z\times\tan\alpha - (R_x \times R_y \times \tan\beta/R) \quad (4.38)$$

$$t\_newy = t0y + ((R_x{}^2 + R_z{}^2)\times\tan\beta/R) \quad (4.39)$$

$$t\_newz = t0z + R_x\times\tan\alpha - (R_z \times R_y \times \tan\beta/R) \quad (4.40)$$

The previous calculations are shown in C++ and OpenGL code as follows:

```
alpha_angle = atanf((2*(image_index+1)-n_pixels-1)/(2*focal_length_norm));
beta_angle = 0.0;// When using a cylindrical lens array
beta_angle  = atanf((-2*(image_index_y+1)-n_pixels-1) /
(2*focal_length_norm));// When //using a spherical lens array
r_x = t_zerox - centre_x;
r_y = t_zeroy - centre_y;
r_z = t_zeroz - centre_z;
```

```
r_xyz = sqrt (r_x*r_x + r_y*r_y + r_z*r_z);
t_newx = t_zerox - r_z*tan(alpha_angle) - ((r_x*r_y*tan(beta_angle))/r_xyz);
t_newy = t_zeroy + (((r_x*r_x + r_z*r_z)*tan(beta_angle))/r_xyz);
t_newz = t_zeroz + r_x*tan(alpha_angle) - ((r_z*r_y*tan(beta_angle))/r_xyz);
core::matrix4 matr;
matr.buildProjectionMatrixOrthoLH(200, 150, -200, 5000);
vector3df(t_newx,t_newy,t_newz));
```

For example, if the camera location was at (-50,82,-13), and the original target point at (0,20,300), vertical rotation angle β = 0º, and the horizontal rotation angle equals to -18º. The cordinates of the new target point will be:

t_newx = - $R_z \times \tan\alpha$ = *-101.69, t_newy = 20, and t_newz =300* + $R_x \times \tan\alpha$ = *283.75*

4.4 The image plane rotation angles calculation

Figure 4.4 shows a cross section of the system when either a micro lens array or cylindrical lens array is used. In the case of perspective projection, the incident rays hitting the pixels in the image plane are shown as well as the rotated micro lens array in 5 different positions. The incident ray that intersects the image plane provides a value to the pixel it hits. If the incident ray is traced back to the scene, and the ray intersects an object at a point, it is possible to consider the value of the pixel it hits is the perspective projection of the object point on the image plane. Therefore, if the same ray intersects any other image plane, the pixel it hits will record the same value of the pixel it hits in the perspective projection. Based on this principle, if each ray is received by another image plane and the value of the pixel it hits is mapped to the *FII*. The image that results from this process is equivalent to the perspective projection of the scene. The reason is that the pixel values of the final image are the same as the pixel values of image rendered with a perspective projection of the scene through a pinhole array or a micro lens array taking into account the characteristics of the micro lens array that affect the incident rays. Each group of parallel rays in the perspective projections is supposed to be received by the image plane of the camera when the camera is rotated with a specific rotation angle. The rotation angles of the camera and the locations of the camera target are selected so that the image plane is perpendicular to each group of the parallel rays apart, and then the orthographic projection is applied to the scene. Specific pixel values are selected from each orthographic image and mapped to the corresponding pixels in the *FII*.

The objects illustrated in Figure 4.4 simulate the real life objects. The scene is created in a graphic application such as 3-D Max, Maya, Blender, etc… and exported to the Visual C++ application software that implements the method of creating integral images. The locations of the camera target, and the rotation angles, as well as the selected and mapped pixel locations are calculated in the light of the characteristics of the used devices including the micro lens array or the pinhole array. In the following discussion, a simplified system is considered in which a pinhole array is used. The rules applied when a cylindrical lens array is used are the same as the rules applied in the case of micro lens array, however, the camera is rotated vertically and horizontally in the latter case, whereas, the camera is rotated horizontally in the former case.

Referring to Figure 4.4, if the following conditions and parameters are considered:
A cylindrical lens array is used. In order to simplify the calculations, a virtual cylindrical pinhole array is considered to replace the practical cylindrical lens array (lenticular array). The vertical pinhole array behaves, as a cylindrical lens array with a focal length equals to the distance between the pinholes and the image plane. Rays' refraction and the effect of the lenses on the rays' direction are neglected to simplify the calculations.
Width of pixel p is measured with a length unit.
Pitch of lens Pt_l is measured with a length unit.
The normalised value of Pt_l is: $Pt = Pt\_l/p$, where p is the width of pixel, Pt is an absolute value with no dimension or unit.

Normalising these values or dividing them by the width of pixel is an approach to use the width of the pixel as a unit of length in the calculations to make them easier and independent of the width of pixel in the various types of screens, and independent of the types of lens arrays that have various focal lengths. The same approach will be applied for other variables measured with the length unit.

The focal length of the lens is Fl measured with a length unit but the normalised value of the focal length F is the actual value divided by the width of pixel, then $Fl = F\times p$.

Width and height of the image plane or the screen are *wl* and *hl* respectively. The normalised values of the width and height are *w* = *wl/p* and *h* = *hl/p* respectively. In this case, w is the number of the horizontal pixels, whereas, h is the number of vertical pixels in the display screen.

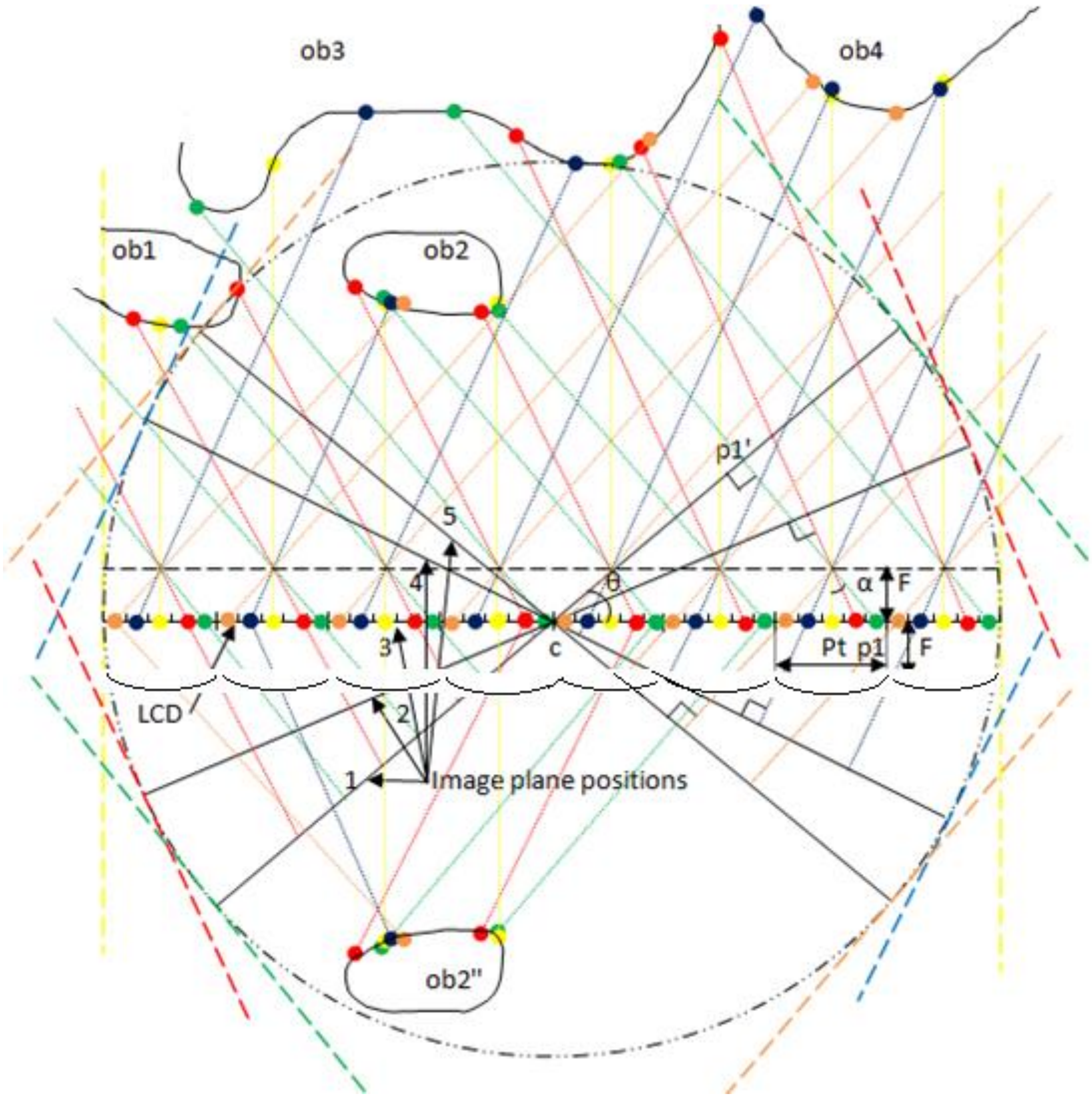


Figure 4.4: Orthographic projections at different rotation angles in the pickup stage, and the objects reconstruction in the replay stage.

The number of horizontal pixels under a lens with the width of *Pt* is *n*, then *Pt* = *n*×*p*.
N is the total number of effective lenses covering the image plane in the capture stage or the screen in the replay stage, in this simple example *n* = 5, *N* = 8.

The pixels under a single lens starting from right to left are *p1, p2, p3, p4,* and *p5*, whereas, the primary positions of these pixels when the image plane is rotated with specific angles are *p1', p2', p3', p4'*, and *p5'* respectively.
An index m is used to indicate the positions of the image plane, from Figure 4.4, the image plane numbered with *1* is indicated with $m = 1$, image plane number *2* is indicated with $m = 2$, and so on.

The number of horizontal pixels under a lens with the width of *Pt* is $n\times Pt = n\times p$.
N is the total number of effective lenses covering the image plane in the capture stage or the screen in the replay stage, in this simple example *n* = 5, *N* = 8.
The pixels under a single lens starting from right to left are *p1, p2, p3, p4*, and *p5*. While the primary positions of these pixels when the image plane is rotated with specific angles are: *p1', p2', p3', p4'*, and *p5'* respectively.
An index *m* is used to indicate the positions of the image plane, from Figure 4.4, the image plane numbered with *1* is indicated with *m = 1,* image plane number *2* is indicated with *m = 2,* and so on.
Point *c* is the centre point of the image planes that are rotated to make different angles with the original image plane that has the angle *0* to which all the pixels are mapped.
The variable $image\_index$ is devoted to indicate the different projections starting with 0 for $m = 1$.
For the first pixel $p1$ with the green colour, the angle α is the angle between the original image plane and the rotated image plane, the approximate value is:

$$tan\,\alpha = p1p1'/c\,p1' = (Pt/2)/Fl = Pt/2Fl \qquad (4.41)$$

A pixel centre is the point on the rotated image plane where the relevant object point in the scene is projected. The incident ray that intersects the rotated image plane in pixel *A* is the ray that starts from the orthogonally projected object point, and then virtually passes through the microlens lens array or the cylindrical lens array and intersects the original image plane in pixel *B*. Therefore, the value of pixel *A* is the same as the value of pixel *B*. For more accuracy, the pixel centre point is considered as the point of the screen where the corresponding point in the scene is projected on the image plane.

If the normal of the image plane makes a negative angle with the original normal, the image plane positions will be represented with the image planes numbered *1* and *2*. If the normal makes an angle of *0*, the image plane position will be represented with the image plane numbered *3* which is the same as the original position, whereas, if the angles are positive, the positions will be represented with the image planes numbered *4* and *5*. Pixels from the resulting image on the image plane with the position *1* will be mapped to the pixels of the vertical columns located under the cylindrical lenses starting from the right hand side of the lenses. The first column from the right side is *p1*; the second is *p2* and so on. If the centres of the pixels are considered, for each image plane position with the index m and the rotation angle α, we can write:

$$tan\,\alpha = \frac{\frac{p}{2}\times(2m-n-1)}{Fl} = p\times(2m-n-1)/2Fl \qquad (4.41)$$

$$\alpha = \tan^{-1}[(2m-n-1)/2F] \qquad (4.42)$$

If the images taken (i.e. the projections of the scene) are indexed with the index called *image_index*. For example, the first projection is given the number *m = 1*, and *image_index = 0*, then *m = image_index +1*, and the last Equation will be:

$$\alpha = \tan^{-1}\{[2((image\_index)+1)-n-1]/2F\} \qquad (4.43)$$

In the example shown in Figure 4.4, *n = 5* and *F = 3*. When $image\_index$ takes the values *0, 1, 2, 3,* and *4,* the set of angles are respectively: *-33.69, -18.43, 0, 18.43,* and *33.69.*

The index m can be assigned with the number of the projection in OpenGL. Each projection is taken when the image plane is rotated with a specific angle α, from each resulting image we need to extract some of the columns that are corresponding to the set of columns on the final image. These columns of pixels are selected based on the equivalence between the values of these pixels and the values of the pixels in the final image that would have the same values in the case when a set of vertical pinholes or a vertical lens array is used to capture the image.

For example, in Figure 4.4, if a lens array or a set of vertical pinholes are used, the column of pixels that has the point $p1'$ in the rotated image plane will have the same pixels values of the column in the final image plane to which the point $p1$ is belonging. Point $p1$ is the location where a point from the scene is projected on the image plane that has a rotation angle of 0. If the image plane is rotated with the angle α, the projection of the same point will be located at $p1'$ from this image plane.

Calculating the distance from the centre of the image plane that is rotated with a zeroed angle to the location where the columns of pixels are mapped is required. If the cylindrical lens array is used, the locations of the columns are the locations on the original image plane where the equivalent columns from the rotated image planes are copied. We need to calculate these locations and write the pixels on them, the values of these pixels are extracted from the images that result from the orthogonal projections on the rotated image planes. The corresponding locations in the rotated planes are calculated by multiplying the distances between the central columns in the image planes to the pixels locations by the cosine of the rotation angle. The mapped pixels form a final image. The final image is equivalent to the perspective projections of the scene on the image plane so that each micro image represents a perspective projection of the scene. In this case, perspective projections of the scene are formed from orthographic projections.

A new variable called *lens_index* is considered so that *lens_index* indicates the indexes of the elemental images that are supposed to be mounted by the lenses in both the capture and the replay stages. *lens_index* starts with 0 to indicate the elemental image at the left end of the screen or the image plane. The maximum value of $lens\_index$ is $\frac{w}{n}$ -1. The distances will be measured with the unit of pixel and considered as the distances between the centres of the pixels. For the final image, the distance between the centre of the image plane and the columns that are copied from the same projection can be found with the following formula:

$$cp_m = \{\frac{w}{2} - [((lens\_index) + 1) \times n - m]\} - 0.5 \qquad (4.44)$$

$$cp_m = \{\frac{w}{2} - [((lens\_index) + 1) \times n - ((image\_index) + 1)]\} - 0.5 \qquad (4.45)$$

For the example shown in Figure 4.4, when $image\_index$ is *0*, and $lens\_index$ varies from *0* to 4.23, the distances from c to the mapped locations (for example *p1*) will be: *15.5, 10.5, 5.5, 0.5, -4.5, -9.5, -14.5,* and *-19.5* respectively. These are the pixels with the green colour.

If the cylindrical lens array is used, the pixels in the screen will be dealt with as columns numbered from *0* on the far left of the screen and increasing rightward. Then it is possible to consider cp_m as the horizontal location of the column combined with specific *image index* and *lens index*. The number of such pixels column is:

$$column_m = ((lens\_index) + 1) \times n - ((image\_index) + 1) \qquad (4.46)$$

Formula (4.45) is used to calculate the locations on the final image or screen for specific $image\_index$ and $lens\_index$ to determine where the pixels should be mapped. Now it is needed to extract the mapped pixels for specific $image\_index$ and $lens\_index$ from the image that is resulting from the projection on a rotated image planes. Locations of points such as *p1'* from the image plane that is rotated with an angle α can be found for each $image\_index$ and $lens\_index$ as follows:

$$cp1' = cp1 \times cos\ \alpha \qquad (4.47)$$

For each ($image\_index$) and ($lens\_index$), the distance from the centre of the screen to the point cp'_m where the columns of pixels should be extracted and mapped to the correct location in the final image is calculated From Equations (4.45), and (4.47):

$$cp'_m = cp_m \times \cos\alpha \qquad (4.48)$$

$$cp'_m = \{\{\frac{w}{2} - [((lens\_index) + 1) \times n - ((image\_index) + 1)]\} - 0.5\} \times \cos\alpha \qquad (4.49)$$

If the cylindrical lens array is used, the columns of pixels are numbered starting from *0* on the far left of the screen and increasing rightwards. It is possible to consider cp'_m as a horizontal location of the column combined with specific ($image\_index$) and ($lens\_index$). The number of such column of pixels is the integer part of $column'_m$

$column'_m = \frac{w}{2} - \ cp'_m$ (4.50)

From Equations (4.49), and (4.50):

$column'_m = \frac{w}{2} - \ \{\{\frac{w}{2} - \ [((lens\_index\ ) + 1) \times n - ((image\_index) + 1)]\}$
$- 0.5\} \times \cos\alpha$ (4.51)

$column'_m = \frac{w}{2} - \ \{\frac{w}{2} \times \cos\alpha - \ [((lens\_index\ ) + 1) \times n - ((image\_index) + 1)] \times \cos\alpha$ -
$0.5\cos\alpha\}$ (4.52)

$column'_m = \frac{w}{2} - \frac{w}{2} \times \cos\alpha + (lens\_index\ ) \times n \times \cos\alpha + n \times \cos\alpha - (image\_index) \times$
$\cos\alpha - \cos\alpha + 0.5\cos\alpha$ (4.53)

$column'_m = \frac{w}{2} - \ [\frac{w}{2} - \ n \times (lens\_index\ + 1) + image\_index + 0.5] \times \cos\alpha$ (4.54)

Formula (4.54) is used to calculate the locations of columns on the image plane or screen for specific $image\_index$ and $lens\_index$. These columns of pixels should be extracted and mapped to the final image. The exact column numbers starting from the left hand side of the screen are the integer part of $column'_m$ which varies in relation to $image\_index$ and $lens\_index$. In fact, the values calculated with Equation (4.54) represent the vertical projection of the corresponding pixels of the final image on their original image plane. Therefore, the locations of these projections determine the locations of the corresponding pixels we need to extract.

In the case of using cylindrical lens array to capture and display, Y coordinates of the pixels on the screen are the same for the extracted pixels and the pixels to which the extracted pixels are mapped to the final image. For this reason, it is possible to map the pixels without change to Y coordinates; in other words, the pixels are shifted horizontally from one position to another. If a microlens array is used to produce a full parallax image, the pixels will be shifted horizontally as well as vertically because X coordinates and Y coordinates of the extracted pixels are different to their coordinates in the final image after mapping. Under each lenslet, there are n columns of pixels of the final integral image in which the columns are grouped in groups of n columns each. Using Equation (4.49) the columns of pixels are selected and extracted from the orthographic images of the view volumes taken at different rotation angles, whereas the rest of the columns are discarded.

On the final integral image, the first column from the left hand side is indexed with 0 for the cylindrical lens array. For the spherical lens array, the upper left pixel of the screen has the coordinates $(x, y) = (0, 0)$ which is the origin of the screen coordinates, and the index 0. The pixel indexes increase gradually when the pixels are scanned horizontally, therefore, the first left pixel from the second upper horizontal line equals to the number of horizontal pixels in a line and so on. The index of each column *x_pixel* can be calculated for *column_index* and image index *image_index* with the Equation: *x_pixel* = *n*×*column_index* + *image_index*, where *column_index* is the index of the group of columns to which the extracted columns should be mapped. With the application software, the Red, Green, and Blue (RGB) values of the pixels are saved in an array of $3\times w\times h$ elements, where w, and h are the width and height of the screen respectively. The indexes of a pixel with x coordinate *x_pixel*, and y coordinate *y_pixel* within the single dimension array of elements are *pixel_ind* , *pixel_ind* + *1*, and *pixel_ind* + *2* for R, G, and B respectively where *pixel_ind* = *x_pixel*×*3* + *w*×*y_pixel*×*3*. For example, in the example shown in Figure 4.4, the angle calculated for the first projection which is -33.69, *image_index* = *0*, the values of $column'_m$ for all the lenses from *lens_index* = 0, to *lens_index* = 7 can be calculated. From Equation (4.54), these are 7.135, 11.285, 15.435, 19.585, 23.735, 27.885, 32.035, and 36.185, these distances starting from the left hand side of the screen will hit the columns of pixels that are marked with the numbers 7, 11, 15, 19, 23, 27, 32, and 36 respectively. After carrying out the orthogonal projections on different image planes with different rotation angles, the selected pixels from the resulting images are mapped to the final image that is the required integral image or the integral image before extra processing. This image is able to replay the scene using a spherical or cylindrical lens array providing the replay device is similar to the virtual device that has been used for the simulation process of capturing the scene.

In the following example, a scene made by Irrlicht will be displayed with OpenGL and replayed using cylindrical lens array with a focal length Fl = 3 mm, the number of pixels under each lens is n = 9, Pt = 2.12 mm, p = 2.12/9 = 0.236 mm, F = 3/p = 12.74, and

$image\_index$ = 0, 1, 2, 3, 4, 5, 6, 7, or 8. Table 4.1 represents the different values of α and t_new when $image\_index$ varies from 0 to 8:

$image\_index$	α	t_newx	t_newy	t_newz
0	-17.43	98.27	0	-15.7
1	-13.25	73.7	0	-11.77
2	-8.92	49.13	0	-7.85
3	-4.49	24.58	0	-3.93
4	0	0	0	0
5	4.49	-24.58	0	3.93
6	8.92	-49.13	0	7.85
7	13.25	-73.7	0	11.77
8	17.43	-98.27	0	15.7

Table 4.1: Values of α and t_new in relation to $image\_index$.

If the employed lens array is cylindrical lens array, the columns of pixels are mapped in different ways. One of these is the one shown in Figure 4.4. In this way, the columns of pixels can be mapped so that the objects appear in front of the screen as real pseudoscopic integrated images when they are reconstructed and replayed. In this type of mapping, the columns of pixels are selected from the images of the view volumes taken at different rotation angles, the columns are selected in dependence to Equation (4.54).

Referring to the previous example, starting from the initial position of the image plane with the rotation angle of α = -17.43, and *image_index = 0*, a set of pixel columns will be extracted from the image of the view volume that results from an orthographical projection on the image plane, whereas, the rest of the columns will be discarded. The number of columns of pixels under each lens in that example is *n* = 9, therefore, if the columns in the image were grouped in groups of 9 columns each, the selected columns will be the set formed with the first column from each group. Similarly, the columns of the final image are grouped to the same number of groups with the same number of columns in each group, and then each extracted column of pixels from the previous image in accordance to Equation (4.54) is mapped to the same locations in the final image. For example, the first extracted column is placed in the column numbered with *0* of the final image, which is the first column on left; the column numbered with 9 is placed on the 9th column of the final image and so on. The numbers of these columns *x_pixel* can be calculated for each column index and image index with the following Equation:

$$x\_pixel = n \times column\_index + image\_index \tag{4.55}$$

where *column_index* is the index of the group of columns to which the extracted columns should be mapped. For the first image in this example:

$x\_pixel = 9 \times column\_index$, or, $x\_pixel = 0, 9, 18, 27,$ etc.

The same way is applied for the images with image indexes *image_index.*
$image\_index = 1, 2, 3, 4, 5, 6, 7,$ and 8.
The upper left pixel of the screen has the coordinates $(x, y) = (0, 0)$ which is the origin of the screen coordinates. x, and y coordinates increase when moving to the right and down respectively.

The RGB values of the pixels are saved in an array of 3×W×L elements, where W, and L are the width and height of the screen respectively measured with pixels. The indexes of the pixel with x coordinate *x_pixel,* and y coordinate *y_pixel* within the single dimension array of elements are *pixel_ind* , $pixel\_ind + 1$, and $pixel\_ind + 2$ for R, G, and B respectively where:

$$pixel\_ind = x\_pixel \times 3 + W \times y\_pixel \times 3 \tag{4.56}$$

For groups of 9 columns, if we want to rotate the pixels around the centre of the cylindrical lenses, *pixel_ind* becomes:

$$pixel\_ind = 3 \times x\_pixel + 24 - 6 \times image\_index + W \times y\_pixel \times 3 \tag{4.57}$$

In general, the z coordinates of the target point must be normalized or divided by a distance value equals to the depth of image so that the target coordinates are independent of the dimensions of the scene. For this reason, a variable must be introduced to divide the z coordinate of the target point, and the software used should be able to tune this variable so that the resulting image quality is optimized.

With the technique of *DCTarget* implemented by the application software, the physical equipment needed in the capture stage is avoided, also, the main disadvantage of integral imaging technique is overcame that is the need for two stages to render the image. Two stages are required to render an integral image because the first stage produces a pseudoscopic image and the second stage is needed to correct the image and convert it to an orthogonal image. Each virtual micro lens images the scene from a different location, points from the scene are projected and recorded in different elemental images, and then these micro-images are replayed and the ray bundles in the replay space are intersected forming images so that these images are changeable in respect to the viewer location. If the viewer location is changed, a new integrated image is automatically formed by a different group of ray bundles. The previous calculations of pixels are shown in C++ and OpenGL code:

```
_cosine_alpha = cos (alpha_angle);
x_pixel = n_pixels*column_index + image_index ;
__x_pixel__ = (floor_)*n_pixels/2 + image_index - n_pixels*(column_index+1);
real_x_pixel = ((floor_)*(n_pixels/2))-(__x_pixel__*_cosine_alpha);
real_x_ = (int)(real_x_pixel);
myobjectscolor = image->getPixel(real_x_, y_pixel);
pixel_ind = x_pixel*3 + Screen_w_pxl*y_pixel*3;// screen_w_pxl=1024
pixels_index[pixel_ind] = myobjectscolor.getRed();
pixels_index[pixel_ind+1] = myobjectscolor.getGreen();
pixels_index[pixel_ind+2] = myobjectscolor.getBlue();
pixel_ind = column_ind*3 + Screen_w_pxl*row_ind*3;
image->setPixel(column_ind, row_ind,
video::SColor(255,pixels_index[pixel_ind],pixels_index[pixel_ind+1],
pixels_index[pixel_ind+2]), false);
```

4.5 The reconstruction stage

After carrying out the orthogonal projections of the scene with different rotation angles of the image plane and different camera targets, the selected pixels from the resulting images are mapped to the final image plane to be used to replay the scene using a spherical or cylindrical lens array with appropriate characteristics. The pixels of the integral image are mapped as described in the pickup stage; the 2-D elemental images are displayed on a screen such as liquid-crystal display (LCD) screen.

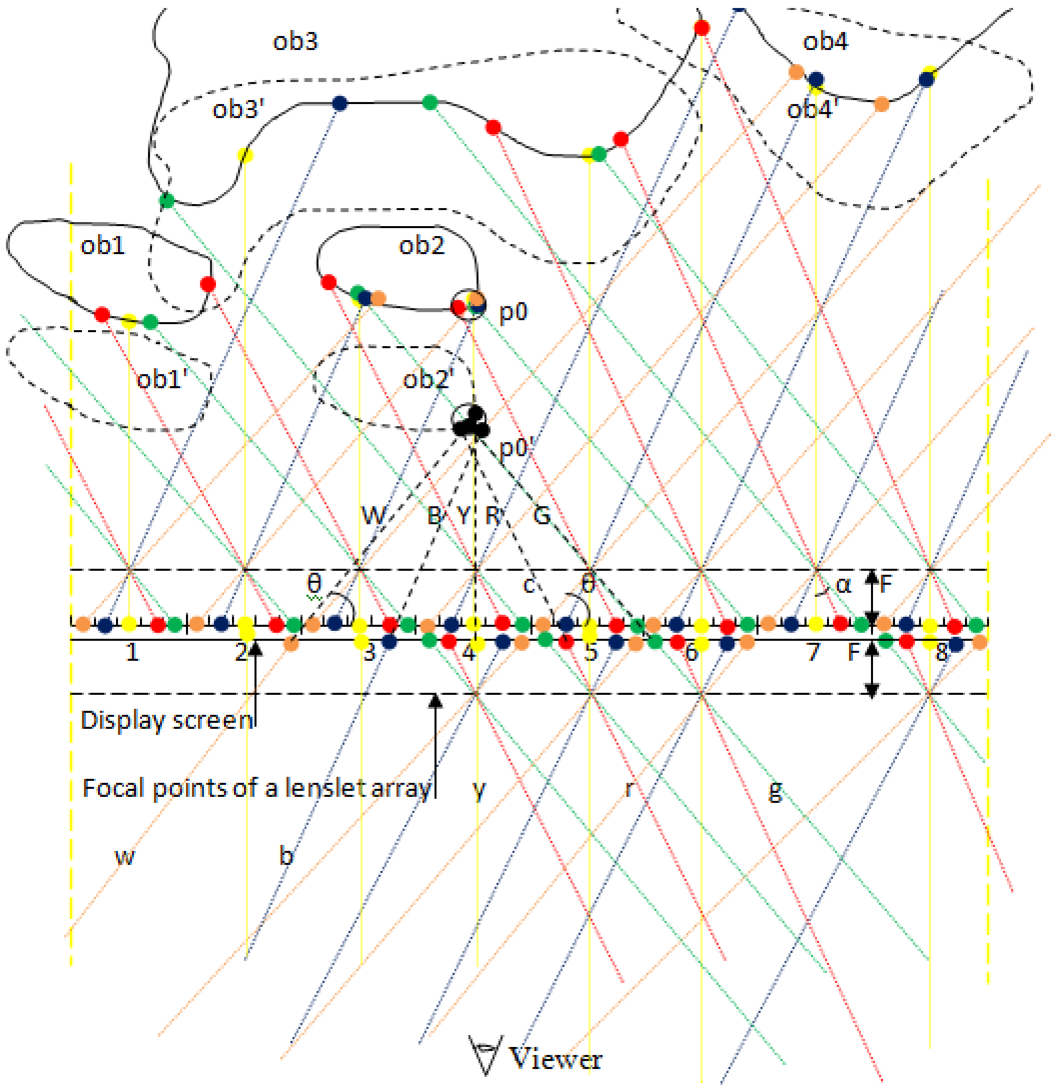


Figure 4.5: The orthoscopic virtual reconstruction stage.

The lens array can be placed on the screen so that the lenslets are precisely covering the elemental images; each lenslet covers one elemental image. If the pixels were mapped in the same order of the sequence of projections on the image plane, and the lenslet array used in the reconstruction stage has the same pitch and focal length of the pickup stage, the reconstructed image will be pseudoscopic real image, or depth reversed image. The viewer sees a different portion of the scene through each cylindrical lenslet or microlens. The integral image is composed of elemental images or portions of the scene. Each portion is composed of pixels selected from different orthographic projections of the scene. Each composed portion is equivalent to the elemental image that is formed under each lenslet or microlens when *DIVGL* method. The elemental images f the scene formed with *DIVGL* method are the perspective images taken by each lens of the lens array. When the integral image is replayed with the lens array, the rays emitted from the pixels are intersected and the portions of the scene are spatially integrated and reforming the scene.

The viewer sees with each eye the integrated scene from a different angle. Each eye receives rays emitted from pixels belonging to the same rotated image plane (i.e. the same orthographic projection on a plane with a rotated camera target). In other words, the viewer sees the integrated scene with each eye from a different angle. Therefore, the viewer is able to see the image with the required 3-D effect, or the required 3-D integrated image.

Figure 4.5 shows the reconstruction process using a lens array with the same focal length F and the same pitch P of the virtual lens array that was employed in the pickup stage. The gap between the surface of the LCD and the focal points of the microlenses or cylindrical lenslets is adjusted to be equal to the virtual focal length.

Light is emitted from the pixels of the LCD screen. The rays proceeding from the pixels that are forming the image displayed on the LCD screen and traversing the microlens array are intersected in points in front of the lens array. The intersection points of the rays in the space in front of the display screen are located on spacial points with proportional distances to the screen equal to the proportional distances between the same object points in the model and the virtual image plane. Figure 4.5 illustrates few replayed objects and their intersected rays.

For example, object *ob2* is reconstructed by the rays proceeding from the LCD and intersected with each other. The intersection points of object *ob2* (*ob2'*) are closer to the screen than the intersection points of object *ob3* because object *ob2* is closer to the image plane than ob3, therefore, the reconstructed points of object ob3 will be closer to the viewer’s eyes than the reconstructed points of object ob2. The image *ob2'* of ob2 will look convex, whereas, in reality, ob2 is concave and it should be displayed so that it looks as it is in reality. The image is formed with converging rays of light.

In order to reconstruct the objects as an orthogonal virtual image we can reverse the columns of pixels in each elemental image around the central vertical column of the cylindrical lenslet. When the microlens array is used, the pixels under each microlens should be rotated 180^o

around the centre of the microlens. Figure 4.5 shows the pixels that were represented in Figure 4.4 after rotating these pixels around the centre of the cylindrical lenslet of 5 columns.

Objects in the scene such as *ob2* are reconstructed to orthographic virtual images *ob2'*. The reconstruction process happens because of the intersection between the virtual extensions of the rays proceeding from the pixels on the display screen and passing through the lenslets used for reconstructing the images. When the viewer is looking on the screen via the lenslet array or the microlens array, the objects are seen by the viewer as virtual images located behind the screen.

The proportional distances of the reconstructed objects to the display screen are equal to the proportional distances of the same objects in the imaged scene to the image plane that was used to capture the images. For example, Figure 4.5 represents the point *p0* from the object *ob2* in the scene; *p0* was captured by the lenses or the pinholes that are forming the elemental images 2, 3, 4, 5, and 6. If the lens array used in the capture stage was cylindrical lens array, the rays *w, b, y, r,* and *g* shown with the colours brown, blue, yellow, red, and green respectively would represent the rays ended at the viewer's eyes passing through the focal points of the lenses, the pixel columns of the mapped pixels where the rays start. If the rays are extended, these would be intersected in virtual points and form virtual image of the scene. As a cylindrical lens array is used in this example, the point *p0* represents a vertical portion of the scene. The virtual extensions of these rays are respectively *W, B, Y, R,* and *G*. If the lenslet array was a microlens array, the point *p0* would represent a point which is projected into the image plane and displayed with a pixel in each related elemental image. The virtual extensions intersect with each other at the point *p0'*. The point *p0'* looks as if it is locating behind the screen. The proportional depth of points such as *p0'* is comparable to the proportional depth of their original points such as *p0* in the scene. In the same way, all the object points in the scene are reconstructed.

It is possible to consider the distance between the constructed virtual point and the surface of the display screen as the depth of that point. *Yl* is the numerical value of the depth of the reconstructed point *ob2'*. *Yl* can be linked to the location of the pixels that hold information about the projections of this point such as the pixel coloured with green in the elemental image number 6 shown in Figure 4.5. For example, the green pixel in the *6th* elemental image holds the colour value of the projection of *ob2* on the image plane that was rotated with the angle α, where:

$$\alpha = \tan^{-1}\{[2((image\_index) + 1) - n - 1]/2F\} \qquad (4.58)$$

From Figure 4.5, $\theta = \pi/2 - \alpha$, the normal on the image plane when its rotation angle is α makes the angle θ with the surface of the display screen. If a point was projected on the image plane that is rotated with different rotation angles, its projection will be included in a number of the elemental images. These elemental images are comprised between two elemental images, the first is the elemental image that includes the projection of that point on the image plane with the maximum angle, and the second one is the elemental image that includes the projection of the point on the image plane when it is rotated with the minimum angle. In other words, the lenslets that are entitled to capture the point are comprised between the two lenslets that capture the point in the cases of rotating the image plane with the maximum angle and the minimum angle. The pinholes or the lenslets that are outside these two lenslets are unable to receive the orthogonal projection of this point.

If the number of these lenslets is μ, their width will be $\mu \times Pt\_l$, where Pt_l is the width of the elemental image or the pitch of the lenslet. It is not necessary that each of these lenslets to include information about the point in question as only specific pixels of each projection of the scene are picked up while the remainders are discarded. The numerical value of the depth Yl can be expressed as:

$$Yl = c \times (\mu \times Pt\_l/2) \times \tan \theta_0 \qquad (4.59)$$

where c is a constant value used to express the relation between the distances on the surface of the image plane and the virtual depth of each point in the scene (i.e. the distances between the virtual reconstructed points and the surface of the screen). θ_0 is the maximum value of the angle θ. The same value of Yl can be obtained by considering any other case of the angle θ and the corresponding length on the surface of the screen instead of $\mu \times Pt\_l$. For a specific value of the angle θ (i.e. θ_0), $\tan \theta_0$ will be constant and the pitch of the lenslet Pt_l is constant, therefore, the numerical value of Yl can be written as:

Yl = C×μ (4.60)

where C = c×($Pt\_l/2$)×$tan\ \theta_0$, and μ is the maximum number of lenslets that can capture the point through the orthographical projections of the scene on image planes with the different rotation angles. In the previous calculations, the effect of the lenses on the rays such as reflection and refraction were neglected, these effects can be significant. Therefore, the effect of the lenses should be taken into consideration to achieve calculations that are more accurate.

4.6 Displaying the image behind and in front of the display screen

Supposing that a cylindrical lens array is used, the columns of pixels can be mapped in different ways. Figure 4.6 shows two manners of mapping the pixels. In the first manner, the columns of pixels are mapped in a way so that the images of the scene objects are reconstructed to appear replayed in front of the screen as real pseudoscopic integrated images. In this type of mapping, the columns of the image with the index $image\_index$ are mapped to the column number n - $image\_index$. For example, the first set of columns that are extracted from the image with the image index $image\_index = 0$ and the rotation angle α is accommodated in the column number n of each group of the final integral image which is the last column on the right. The second set is extracted from the image with the image index $image\_index = 1$ and placed on the column n-1 of the final integral image and so on.

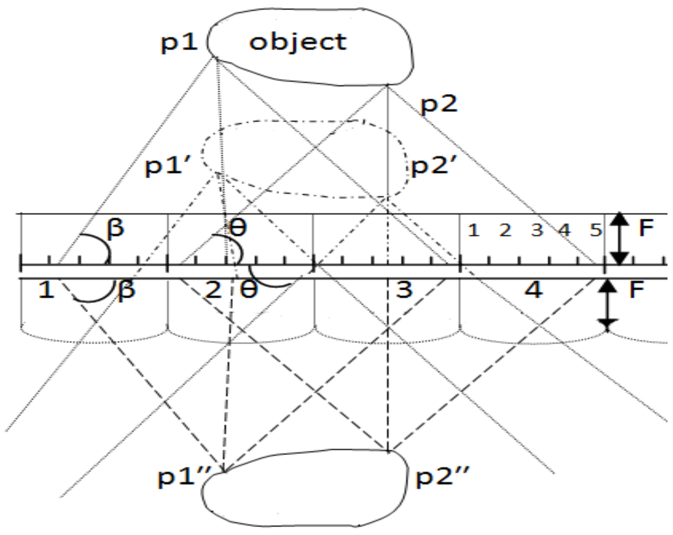


Figure 4.6: Two ways of the objects reconstruction in the replay stage, producing an orthogonal virtual image, and a pseudoscopic real image.

In this way the pixels are mapped in an opposite order of the sequence of projections on the rotated image plane, and the lens array used in the reconstruction stage has the same pitch and focal length of the pickup stage, therefore, the reconstructed image will be pseudoscopic real image, or depth reversed image. Figure 4.6 shows the reconstruction process using a lens array with the same focal length *F* and the same pitch *P* of the pickup stage pinhole array, the gap between the surface of the LCD and the lens array focal points is adjusted to be equal to the virtual focal length. In order to apply the first way, the columns of the first orthogonal image are mapped to the columns *5* and so on. The incident rays proceeding from the pixels on the final integral image that is recorded on the LCD are intersected in points located at proportional distances to the screen. The distances of these points are equal to the proportional distances between the same object points in the scene and the virtual image plane. For example, the illustrated object in Figure 4.6 is reconstructed by the intersection between the rays proceeding from the LCD. The pixel hit by the ray that makes an angle β with the final integral image is replayed in the replay stage as a ray making the angle $-\beta$ with the final integral image. Thus, the object and its replayed image using this way are symmetric about the final integral image plane, i.e. the display screen. Using the mentioned manner, the object points in the scene such as *p2* are reconstructed to points such as *p2''*, the image formed with these converging rays of light is a pseudoscopic real image.

Another manner of mapping the pixels leads to reconstruct the integral image as an orthogonal virtual image; this manner is implemented by reversing the locations of the columns in each elemental image around the central vertical column of the elemental image. When a spherical microlens array is used, each pixel under a microlens should be rotated 180^o around the centre of the elemental image. In Figure 4.6, the pixel hit by the ray that makes an angle θ with the final integral image is shifted about the centre of the elemental image to a symmetric position and replayed in the replay stage as a ray making the angle $180^o - \theta$ with the final integral image. The incident ray and the replayed ray are parallel, therefore the replayed rays are diverged. Using this manner, the image formed with the diverging rays of light is an orthogonal virtual image. The object points in the scene such as *p2* are reconstructed to points such as *p2'*, the image formed with these diverging rays of light is a pseudoscopic real image. The reconstruction process happens because of the intersection between the virtual extensions of the rays proceeding from the pixels on the display screen and passing through the lenses, then, the image is seen by a viewer as an orthogonal virtual image appearing behind the screen.

An orthogonal real image with the possibility to display a part of the image as an orthogonal real image in front of the display screen and the other part as an orthogonal virtual image behind the screen is the favourite form of displaying an integral image. To implement such an image the part in front of the screen of the pseudoscopic real image must be converted to an orthogonal real image, and then displayed on the same final integral image together with the part located behind the screen as an orthogonal virtual image, the final integral image now would be called the new final integral image. The part behind the screen is already projected and its pixels are extracted and mapped to the final integral image with the second manner of mapping. Therefore, the pixel values that represent this part of the scene are mapped to the new final integral image and their values and locations are kept the same as they were on the final integral image. Afterword, the part in front of the screen can be projected and the pixel values of the resulting orthogonal real image of this part are mapped to the new final integral image. The pixels representing the object points of the part behind the screen that are obscured by the object points of the part in front the screen would be overwritten by the pixel values that represent the object points of the part in front the screen.

To convert the part in front of the screen from a pseudoscopic real image, to an orthogonal real image, the camera target is rotated 180˚ around the vertical axis located in the virtual screen and intersecting the centre of that screen. The part in front is now projected on the rotated screen with the same method of the first capture stage. If the lens array used is cylindrical lens array, the pixels of the resulting image are flipped symmetrically around the central vertical axis that is intersecting the centre of the virtual screen. If the lens array is spherical, the pixels are flipped symmetrically around the centre of the screen. The pixel values are written to the new final integral image and some pixels from the second stage overwrite the pixels in the first stage because the part of the scene that is imaged in the second stage obscures the part imaged in the first stage. The final resulting image would be displayed as an orthogonal real image in front of the screen and an orthogonal virtual image behind the screen, which is the favourite scenario of displaying an integral image.

In the replay stage, the part behind the virtual screen is replayed as an orthogonal virtual image, therefore, the pixel representing this part of the scene are flipped around the centre of the elemental images, mapped to the new pixel locations, and their values are kept as they are to form a part of the new final integral image. The part of the scene located behind the virtual screen that is replayed as a pseudoscopic real image in the replay stage is imaged in the same method of the first stage and the pixels are mapped to the new final integral image to form the other part of the targeted integral image. Rendering the part in front the screen can be implemented with the application software by dividing the scene into two parts and moving the image plane z-coordinate to the location where the virtual screen is supposed to be located. The first part is the one behind the virtual screen or the object points of the scene with z-coordinate higher than the z-coordinate of the virtual screen taking into consideration that z-axis is vertical on the screen and directed to the same direction of the viewer i.e. away from the screen. The part behind the screen is projected and the pixels are flipped and mapped as it is mentioned above and the virtual orthogonal image of this part is produced.

In order to produce the real orthogonal image of the part located in front of the screen in which z-coordinate value of the object points are lower than z-coordinate of the virtual screen, the camera target is rotated 180˚ around the vertical axis located in the virtual screen and intersecting the centre of that screen. The camera target is vertical on the screen and directed towards the positive z-coordinates. The part in front is now projected on the rotated screen with the same method of the first stage and the pixels are selected from the orthographic projections and mapped to the image plane so that the replayed scene is a real pseudoscopic image. If the lens array used is cylindrical, the pixel of the resulting image are flipped symmetrically around the central vertical axis that is intersecting the centre of the virtual screen, if the lens array is spherical, the pixels are flipped symmetrically around the centre of the screen.

The pixel values representing the part in front are mapped to the same image of the part behind. Some pixels from the second stage overwrite the pixels in the first stage because the part of the scene that is imaged in the second stage obscures the part imaged in the first stage. The resulting image that is composed form the pixels representing the parts in front as well as the part behind are replayed and both of the parts are replayed so that the part behind the screen is replayed as an orthogonal virtual image and the part in front is replayed as an orthogonal real image.

4.7 Animated integral images

3-D video systems have been pursued for decades as the video format of the future. Various approaches for providing a perceived depth have been invented [11]. Animated films can be made of a series of static integral images using the application software. The images can be viewed using either a spherical lens array or a cylindrical lens array. Animated films can be saved and displayed using special applications.

Animated scenes composed of integral images can be produced with *DCTarget* method. 2-D animation is supposed to be designed as a stream of static images displayed in sequence. A minimum number of static images per time unit should be produced so that flickering is avoided. A rate of *25* frames per second (*fps*) can be enough to prevent flickering [53]. Using the application software, animated scenes can be imported and converted to animated scenes with a 3-D effect. The animated scenes based on sequences of integral images can be named Animated Integral Images (AII). DCTarget algorithm or *DIVGL* is applied to each frame or image apart to be converted to an integral image. A sequence of these integral images can form an AII scene. If a powerful computer is employed so that the conversion form a 2-D animated scene to a 3-D AII scene is fast enough, the conversion can be suitable for real time applications such as video games. Real time conversion can instantly produce 3-D AII scenes from normal 2-D video images. Employing computers with inadequate processing efficiency leads to slow processing, in this case, the application software is suitable for applications where the AII scenes can be rendered slowly ahead of time, such as films and animated advertisement. The application software produces the AII video content and saves the video in a file. The file can be opened with a suitable program such as Windows Live Movie Maker to display the video on a normal PC display screen or a special display screen. The display screen should be mounted with a lenticular or spherical lens array to be able to display the AII video with the required 3-D effect. The application software can be developed to allow the user to display the animated scenes instantly as real time movies. As an example, A short film of about 800 integral images or frames (duration of about 32 seconds) representing several objects was produced with *DCTarget* method to be displayed on a screen supplied with cylindrical lens array. Another similar movie is made to be displayed on a screen using a micro lens array. The film is saved and displayed with Windows Live Movie Maker software.

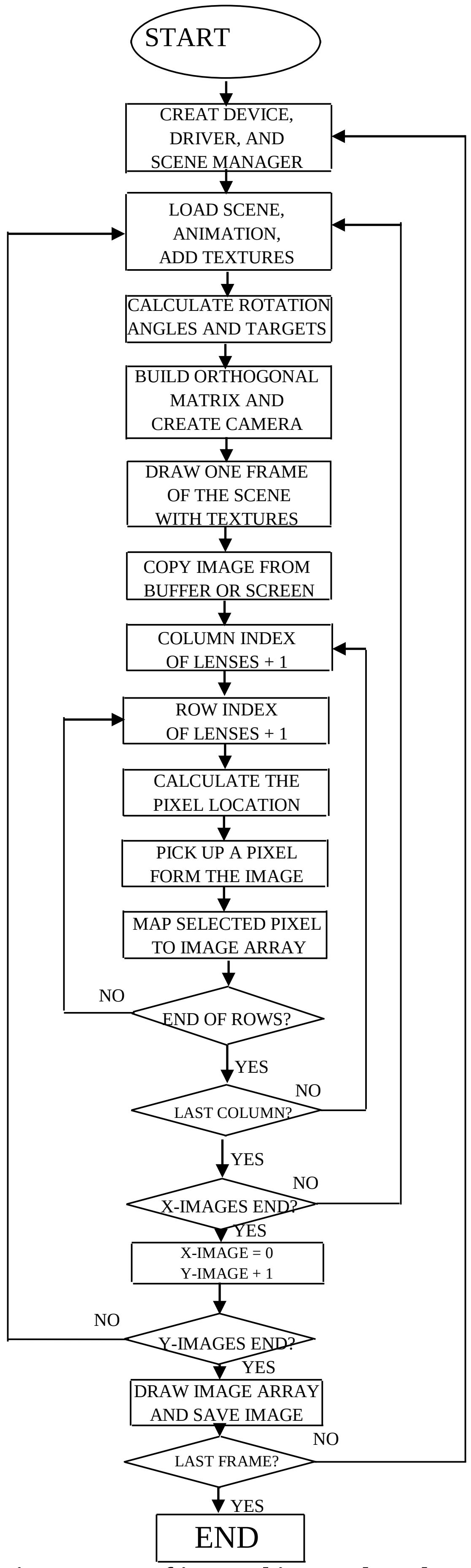

START
CREAT DEVICE, DRIVER, AND SCENE MANAGER
LOAD SCENE, ANIMATION, ADD TEXTURES
CALCULATE ROTATION ANGLES AND TARGETS
BUILD ORTHOGONAL MATRIX AND CREATE CAMERA
DRAW ONE FRAME OF THE SCENE WITH TEXTURES
COPY IMAGE FROM BUFFER OR SCREEN
COLUMN INDEX OF LENSES + 1
ROW INDEX OF LENSES + 1
CALCULATE THE PIXEL LOCATION
PICK UP A PIXEL FORM THE IMAGE
MAP SELECTED PIXEL TO IMAGE ARRAY
NO
END OF ROWS?
YES
NO
LAST COLUMN?
YES
NO
X-IMAGES END?
YES
X-IMAGE = 0
Y-IMAGE + 1
NO
Y-IMAGES END?
YES
DRAW IMAGE ARRAY AND SAVE IMAGE
NO
LAST FRAME?
YES
END

Figure 4.7: The generation process of integral images based on *DCTarget* algorithm.

equal to the number of the rows of pixels under each microlens in the microlens array, the index of the current frame from the animated scene that is projected while the camera target is moving, the current horizontal angle with which the camera is rotated, and the current vertical angle with which the camera is rotated. *pixels_index* array is defined to hold the pixel values of the whole image displayed on the screen. The size of *pixels_index* array is equal to the number of the pixels in the screen multiplied by *3*. Each pixel in the screen has *3* adjacent RGB colours; the *3* RGB colour values are expressed with 3 members of the array so that each member holds the pixel colour value of one of the *3* colours. The size of the array is the width of the screen times the length of the screen times *3*.

Irrlicht device is created and defined to determine the dimensions in pixels of the rendered image and the application programing interface OpenGL. The function *createDevice* returns a pointer *device* to an object from the class *IrrlichtDevice*. In addition, a video driver and a scene manager are defined. The function *getVideoDriver()* which is a member function of the object pointed to by *device* from the class *IrrlichtDevice* is called to define a video driver. *getVideoDriver()* returns a pointer *driver* to an object from the class *IVideoDriver* from the namspace *video* of Irrlicht library. The function *getSceneManager()* which is a member function of the object pointed to by *device* from the class *IrrlichtDevice* is called to define a scene manager. *getSceneManager()* returns a pointer *smgr* to an object from the class *ISceneManager* from the namspace *scene* of Irrlicht library.

The selected scene including the mesh or the geometric structure of the scene, the frames that are forming the animation of the scene, and the texture files. The geometric data about the scene or the nodes forming the scene are saved in files with different formats. The engine is supplied with functions devoted to load the geometric data files and extract the required data to be used in rendering the scene with OpenGL. Variety of formats is supported by Irrlicht engine including 3D Max files and files with the extension .pk3, .DM2, .x and others. The function *getMesh which* is a member function of the object pointed to by *smgr* from the class *ISceneManager* is called to load the geometric data of the scene. The argument of the function *getMesh* is the name of the file that contains the geometric data of the scene. The file is named with its path to provide the function with its location in the computer. The function *getMesh()* gets the mesh from the scene manager. *addAnimatedMeshSceneNode()* adds a scene node to display the mesh. Based on the scene, *getMesh()* returns a pointer to an object from the class *IMeshSceneNode, IAnimatedMeshSceneNode,* or *IAnimatedMesh* from the

Irrlicht namespace *scene.* In order to get the textures and map them to the mesh in the scene, the function *getTexture* which is a member function from the class *IVideoDriver* is used. The argument of *getTexture* is the name and the location of the file that contains the required texture. Textures are saved in image files with different formats such as bitmap images with the extension .bmp, and JPEG images with the extension .jpg.

To handle the animated scenes, the class *IAnimatedMeshSceneNode* includes functions such as the function *setMD2Animation* to set an *MD2* scene as an animated scene, and the function *setAnimationSpeed* to set the speed of the animation or the number of rendered frames per second. The animated scene node or object is pointed to by a pointer returned by a function such as *addAnimatedMeshSceneNode* from the class *ISceneManager*, with this function the animated scene node is loaded from the computer and added to the scene.

The location, the size, and the angular position, of the different objects that are forming a scene are adjusted using Irrlich functions to set the position of the mesh in the scene and the rotation angles. Translation, rotation, and scaling can be applied to the objects to form the required scene. The function *setPosition* that is a member function of the class *IMeshSceneNode* from the namespace *scene* takes the position of the object as an argument. *setPosition* is a function called from a pointer to an object from the class *IMeshSceneNode* which is returned by a function from the class *ISceneManager* such as *addOctreeSceneNode*. A pointer to a scene node object from the class *IAnimatedMeshSceneNode* is returned by the function *setScale* that is applied to the object and takes the vector of x, y, and x scale factors as an argument. *setScale* makes the object seem scaled by the factors x, y, and z, in other words, x, y, and z coordinates of the vertices forming the object in the scene are multiplied by these factors respectively. The function *addLightSceneNode* from the class *ISceneManager* returns a pointer to an object from the class *ILightSceneNode*, this function creates a light scene node and adds it to the scene, the arguments of the function provides the parameters of the light including the location, and the colour.

The integer variable *n_pixels_x* indicates the number of the horizontal pixels under each spherical microlens or cylindrical lenslet when the display screen is covered by a spherical lens array or a cylindrical lens array respectively. *n_pixels_y* indicates the number of the vertical pixels under each spherical microlens when the display screen is covered by a spherical lens array. *focal_length_norm* is the focal length of the lenses that are supposed to display the integral images. *alpha_angle* is the horizontal rotation angle of the camera, and *beta_angle* is the vertical rotation angle of the camera. These angles are calculated with the following two Equations: *alpha_angle = atanf((2×(image_index_x+1)- n_pixels_x - 1)/(2×focal_length_norm)); and beta_angle = atanf((-2×(image_index_y+1)- n_pixels_y - 1)/(2×focal_length_norm));*. *image_index_x* is a variable indicating the current projection of the scene when the camera is horizontally rotated. For each vertical rotation angle, projections are carried out at each horizontal rotation angle when the camera is horizontally rotated. These images are numbered from *0* for the first image on the left hand side to *n_pixels_x – 1* for the last image on the right hand side. *image_index_y* is a variable indicating the current projection of the scene when the camera is vertically rotated. These images are numbered from *0* for the images on the lowest angle to *n_pixels_y – 1* for the image on the highest angle.

t_zerox, t_zeroy, and t_zeroz are real variables to hold the *x*, *y*, and *z* coordinates of the initial position of the camera target. These variables are used to calculate the coordinates of the new target points of the camera. The coordinates of the target's initial position are initialised to select a convenient camera target allowing the viewer to view the scene as required. The user can change these values until they find the required view of the scene. *centre_x, centre_y, and centre_z* are real variables to hold the *x, y,* and *z* coordinates of the camera centre. These variables are used to calculate the coordinates of the camera new target coordinates. The coordinates of the camera centre position are initialised to select a convenient point of view from which the viewer views the scene. These variables are changeable to allow the user to select the required view of the scene. *r_x, r_y,* and *r_z* are the projections of the distance between the initial camera target and the camera centre on *x, y,* and *z* axis respectively. The length of this distance *r_xyz* is calculated and used to calculate the coordinates of the camera target point for each new position of the camera (i.e. new horizontal and vertical rotation angles *alpha_angle* and *beta_angle*) as follows:

r_xyz = sqrt (r_x×r_x + r_y×r_y + r_z×r_z);

t_newx=t_zerox-r_z×tan(alpha_angle)-((r_x×r_y×tan(beta_angle))/r_xyz);

t_newy=t_zeroy + (((r_x×r_x + r_z×r_z)×tan(beta_angle))/r_xyz);

t_newz=t_zeroz+r_x×tan(alpha_angle)-((r_z×r_y×tan(beta_angle))/r_xyz);

In order to build the orthogonal projection matrix, an object *matr* from the class *matrix4* belonging to the namespace *core* of Irrlicht engine is defined. The member function from the class *matrix4 buildProjectionMatrixOrthoLH (f32 widthOfViewVolume, f32 height, f32 zNear, f32 zFar)* builds the matrix for the orthogonal projection. The first argument is the width of the view volume from the space that is intended to be projected; the unit used is belonging to a special type of real variables defined in Irrlicht called *f32* where 32 is the length of the variable value. The third argument is the z-coordinate of the near clipping plane that clips the view volume from the near side to the camera. The fourth argument is the z-coordinates of the far clipping plane that limits the view volume at the far side from the camera. All the four arguments are measured with the same length unit and their values are expressed with real values from *f32*.

The member function *setProjectionMatrix(matr, true)* from *ICameraSceneNode* sets the projection matrix based on calculated factor of the camera such as the coordinates of the centre and target. The pointer *camera* pointing to an object form the class *ICameraSceneNode* belonging to the namespace *scene* of Irrlicht engine is created to define a camera scene node. The member function *addCameraSceneNode* from *ISceneManager* returns a pointer assigned to *camera*. The arguments of this function include the coordinates of the camera centre and the current target point. The device is run with the function *run()* form *IrrlichtDevice,* whereas, the member function *drawAll()* from *ISceneManager* implements projecting and rendering the scene.

aframe_index indicates the current rendered frame from the animated scene. The member function *createScreenShot()* from the class *IrrlichtDevice* copies the data of the rendered image from the buffer and returns a pointer assigned to the pointer *image* that is pointing to an object from the class *IImage* belonging to the Irrlicht namespace called *video*. The data includes the RGB colour data of the rendered image that is saved in the front buffer of the rendering device.

The following piece of C++ code implements the process of reading the different projection and picking up the required pixels from the rendered images:

```
while(device->run())
{
int bframe_index = frame_index % max_anim_frames;
node->setFrameLoop(aframe_index, aframe_index);
```

```
driver->beginScene(true, true, 0);
smgr->drawAll();
driver->endScene();
irr::video::SColor myobjectscolor;
video::IImage* image = device->getVideoDriver()->createScreenShot();
if (image && (image_index_x >= 0))
 {
for (u32 column_index = 0 ; column_index < floor((screen_w_pxl/ lenslet_w_pxl));
column_index++)
      {
   for (u32 row_index = 0 ; row_index < floor((screen_l_pxl/ lenslet_l_pxl)); row_index++)
         {float _cosine_alpha = cos (alpha_angle);
         int x_pixel = lenslet_w_pxl *column_index + image_index_x ;
         //int x_pixel = lenslet_w_pxl *column_index - image_index + lenslet_w_pxl;
         int __x_pixel__ = floor((screen_w_pxl/2))- image_index_x - lenslet_w_pxl
         *column_index;
         float real_x_pixel = (screen_w_pxl/2)-(__x_pixel__*_cosine_alpha);
         int real_x_ = (int)(real_x_pixel);
         //u32 _x_pixel_ = (int)real_x_pixel;
         float _cosine_beta = cos (beta_angle);
         int y_pixel = lenslet_l_pxl *row_index + image_index_y ;
         //int x_pixel = lenslet_l_pxl *column_index - image_index + lenslet_l_pxl;
         int __y_pixel__ = floor((screen_l_pxl/2))- image_index_y - lenslet_l_pxl *row_index;
         float real_y_pixel = (screen_l_pxl/2)-(__y_pixel__*_cosine_beta);
         int real_y_ = (int)(real_y_pixel);
         myobjectscolor = image->getPixel(real_x_, real_y_);
         int pixel_ind = x_pixel*3 + screen_w_pxl *y_pixel*3;
         //int pixel_ind = 3*x_pixel +3* lenslet_w_pxl - 6*image_index + screen_w_pxl
         //*y_pixel*3;
         //int pixel_ind = 3*x_pixel + 3* lenslet_w_pxl - 6*image_index_x + screen_w_pxl
         //*y_pixel*3;
         pixels_index[pixel_ind] = myobjectscolor.getRed();
         pixels_index[pixel_ind+1] = myobjectscolor.getGreen();
         pixels_index[pixel_ind+2] = myobjectscolor.getBlue();
```

```
            }
        }
    }
            if (image_index_x< lenslet_w_pxl) {goto loop1;}
            else {image_index_x = 0;
            image_index_y = image_index_y + 1;
            if (image_index_y< lenslet_l_pxl){goto loop1;}
            else     {goto loop2;}
            }
}
```

max_anim_frames is the number of frames in an animation scene. The frames are rendered in sequence and can be repeated periodically. *setFrameLoop(aframe_index, aframe_index)* sets the current frame as the frame indexed with *aframe_index*. *myobjectscolor* is an object from *SColor* of *video* namespace used to access the member functions *getRed(), getGreen(), and getBlue()* of the class *SColor*. These functions are used to extract the red, green and blue pixel values respectively from a specific RGB pixel.

The FOR loop starts from *column_index = 0* to *floor((screen_w_pxl/ lenslet_w_pxl)),* where *screen_w_pxl* is the number of horizontal pixels of the display screen, *lenslet_w_pxl* is the number of horizontal pixels in each micro-image. The internal nested FOR loop starts from *row_index = 0* to *floor((screen_l_pxl/ lenslet_l_pxl)),* where *screen_l_pxl* is the number of vertical pixels of the display screen, *lenslet_l_pxl* is the number of vertical pixels in each micro-image. For each micro-image with specific *column_index* and *row_index, _cosine_alpha* and *_cosine_beta* are calculated. *x_pixel* is the horizontal index of a pixel from the screen. Each pixel in the screen has an x and y coordinates measured in pixels. *image_index_x* indicates the current image number when rotating horizontally for a specific vertical rotation angle. *image_index_y* indicates the current image number when rotating vertically for a specific horizontal rotation angle. The screen is indexed starting from the pixel on the upper left hand corner of the screen with *x_pixel=0,* and y*_pixel=0.* The screen is scanned downward from left to right, the next pixel in the same row has *x_pixel=1,* and y*_pixel=0* and so on. The first pixel on the left hand side of the second row from the top of the screen is indexed with *x_pixel=0,* and y*_pixel=1 and so on*. Mapping the pixels to the final image affects the 3D effect of the replayed integral image. If the pixels were mapped to the micro-images in reverse order or forward order, the integral image will look behind or in front of the display screen respectively. To select the former or the latter case, the horizontal pixel index will be respectively: *int x_pixel = lenslet_w_pxl ×column_index - image_index + lenslet_w_pxl,* or *x_pixel = lenslet_w_pxl ×column_index + image_index_x.*

In order to pick up the required pixels from the rendered images, the distances from the centre of the screen where the camera centre is located to the location where the pixels should be mapped to the final image are calculated, these horizontal and vertical distances are respectively: *__x_pixel__* and *__y_pixel__*, the distances are measured in pixels. The pixel indexes starting from the far left for horizontal pixels and from the top for vertical pixels are calculated and called respectively *real_x_* and *real_y_*.

After calculating the indexes of the pixels needed to be picked up from the rendered images, the member function *getPixel(real_x_, real_y_)* from the class *IImage* is called. The function *getPixel()* returns the object *myobjectscolor* from *SColor*. The member functions *getRed(), getGreen(),* and *getBlue()* from the class *SColor* return the RGB pixel values of the selected pixel that express respectively the red, green , and blue colour quantity in the pixel. The values are assigned respectively to the red, green and blue colour values of the corresponding pixel in the final image. The index of the target pixel in the final image called *pixel_ind* is calculated. The final image pixel values are saved in a single dimension array. The pixel is expressed with 3 adjacent members in the array, each member of these holds a colour value of the pixel (i.e. red, green, and blue respectively). The pixels are indexed starting from the upper left corner of the screen. The indexes increase when scanning towards the right hand side and downward. Each pixel has 3 indexes for red, green and blue colours with the indexes *pixel_ind* , *pixel_ind +1,* and *pixel_ind +2* respectively, therefore *pixel_ind = x_pixel×3 + screen_w_pxl ×y_pixel×3.*

Once the first projection is done and the pixels are extracted and mapped to the final image, *image_index_x* is checked if reached *lenslet_w_pxl* that is the number of horizontal pixels in a micro-image. If not all the required images during the horizontal rotation are implemented for a specific vertical rotation angle, the angle is increased and the projection is repeated until the end. When the last projection is done, *image_index_y* is increased, the projection started again, the vertical angle is increased, the horizontal angle is zeroed, then the projections are repeated for the new vertical angle until the end of the horizontal angles and so on for all the vertical rotations.

At the end of each projection, the selected pixels are picked up and saved in the array that is called *pixels_index* to be mapped later to the final image. The function *createScreenShot()* returns a pointer to an object from *IImage* that is an image object. The image object is the final image and the pixels of this image are the saved pixel values already saved in the array *pixels_index*. To map these pixel values to the created image, the following loop allows the pixel values to occupy their required location in the new image:

```
for (u32 column_ind = 0 ; column_ind< screen_w_pxl; column_ind++) {
        for (u32 row_ind = 0 ; row_ind < screen_l_pxl; row_ind++){
        u32 pixel_ind = column_ind*3 + screen_w_pxl *row_ind*3;
        image->setPixel(column_ind, row_ind,
        video::SColor(255,pixels_index[pixel_ind],pixels_index[pixel_ind+1],pixels_index[pi
        xel_ind+2]), false); }  }
```

Each pixel from the screen is belonging to specific row and column. For each pixel, the function *setPixel()* is called to set the RGB pixel values saved in the array *pixels_index* to their correct place in the image pixels array. When the image is formed, The member function *writeImageToFile(image, buf, 85)* from *IrrlichtDevice* writes the image formed from the pixels to the buffer, and the function *snprintf()* creates a .jpg image file with a selected name and indexed with the number of the rendered frame to be saved in the folder containing the project. The variable *frame_index* that indicates the current frame in the animated scene is checked, if the scene is a static image, *frame_index* holds a constant value of 1, therefore, one integral image is created and saved for the image, and the algorithm is ended. Otherwise, *frame_index* is increased and the algorithm is repeated for the new frame and a new integral image is created and saved in the project folder or a selected folder and so on until the end of the frames forming the animated scene. The images are submitted to a program to display these images in sequence on a display device supplied with a lens array with identical characteristics and parameters to the ones employed in the mentioned algorithm. If the image is static, the viewer is able to see an image with 3-D effect, while an animated scene with 3-D effect can be viewed when displaying the different frames in sequence.

4.8 Implementations and results

In the following example, a 2-D scene made with 3-D Max and saved as a COLLADA file was used to produce the integral image illustrated in Figure 4.8. Irrlicht engine and OpenGL graphic interface application were used in the environment of C++. The image shown in Figure 4.8 was rendered to be viewed with a cylindrical lens array with a focal length Fl = 3 mm, the number of pixels under each lens is n = 9, Pt = 2.12 mm, p = 2.12/9 = 0.236 mm, F = 3/p = 12.74, and im_in = 0, 1, 2, 3, 4, 5, 6, 7, and 8. Figure 4.9 shows the same scene rendered as a 2-D image before conversion to 3-D integral image. Another example is shown to prove the ability to represent the details of a scene in the resulting integral images. The parameters considered the same as the previous example. Figure 4.10 shows the original 2-D scene rendered using Irrlicht engine without applying *DCTarget* algorithm. The objects shown in the scene are imported from files with different formats.

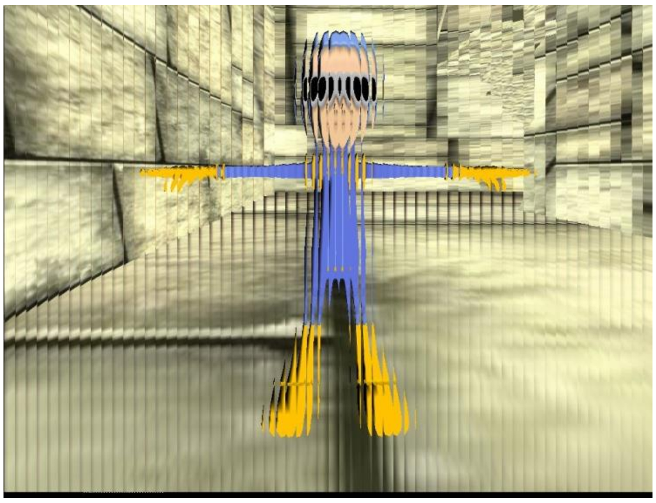


Figure 4.8: An integral image produced by software based on *DCTarget* method.

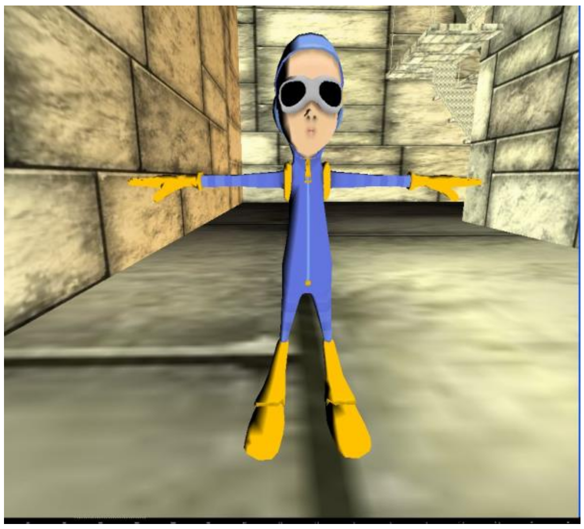


Figure 4.9: The 2-D COLLADA scene before conversion to a 3-D integral image.

The scene shown in Figure 4.10 as a 2-D image is converted to a 3-D integral image to be displayed using a spherical microlens array. Micro-images are displayed on a normal PC screen and each micro-image occupies a square of *3×3* pixels. For more accurate image, the pixels representing the micro-images can be distributed on circles to be covered by the spherical micro lenses. The square micro-images are easier to render, therefore, the image approximates the required one. The number of pixels representing the micro-image is proportionally small because of the limitation of the PC screen resolution. The higher resolution the higher image quality is produced. The examples are meant to display on a PC screen with a resolution of *75 dpi*. Figure 4.11 is the integral image of the scene shown in Figure 4.10.

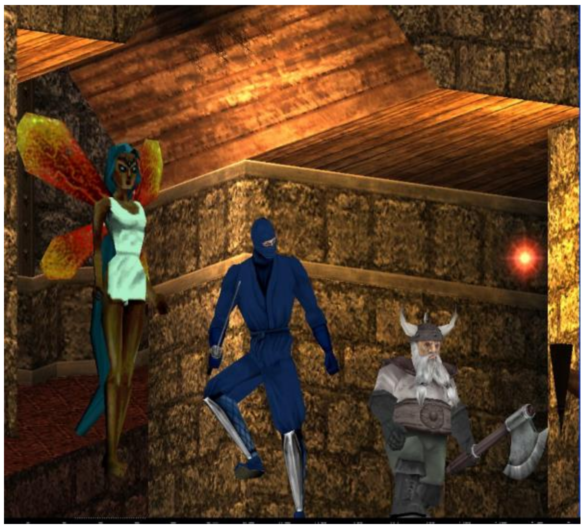


Figure 4.10: 2-D image rendered with Irrlicht engine.

The integral image shown in Figure 4.11 was rendered with vertical rotation of the camera; therefore, using spherical lens array to display the image provides vertical parallax as well as horizontal parallax (i.e. full parallax), whereas, rendering integral images with horizontal rotation only, and displaying these with cylindrical lens array provides only horizontal parallax (e.g. Figure 4.13).

The scene shown in Figure 4.13 is a frame from an animated scene composed from *800* frames. The original 2-D scene is made by Irrlicht and saved in different files with different file formats; each object in the scene was designed and saved in a separated file. The scene is more complicated than the scene shown in Figure 4.9; therefore, this example represents the level of complexity and quality of the integral images produced with *DCTarget* method. If the scene contains objects located at different distance from the camera, and some of these objects obstacle the viewer from viewing parts of the objects behind, parallax in the image allows the viewer to view hidden parts of the objects obscured by other objects. When viewing a full parallax image, moving the viewer vertically or horizontally does not cause significant jumping in the image. However, in some cases, a little jumping can be witnessed due to the low resolution of the display screen. That happens when the resolution does not allow accommodating an adequate number of images in the scene, and therefore, the smoothness of viewing the images is not perfect.

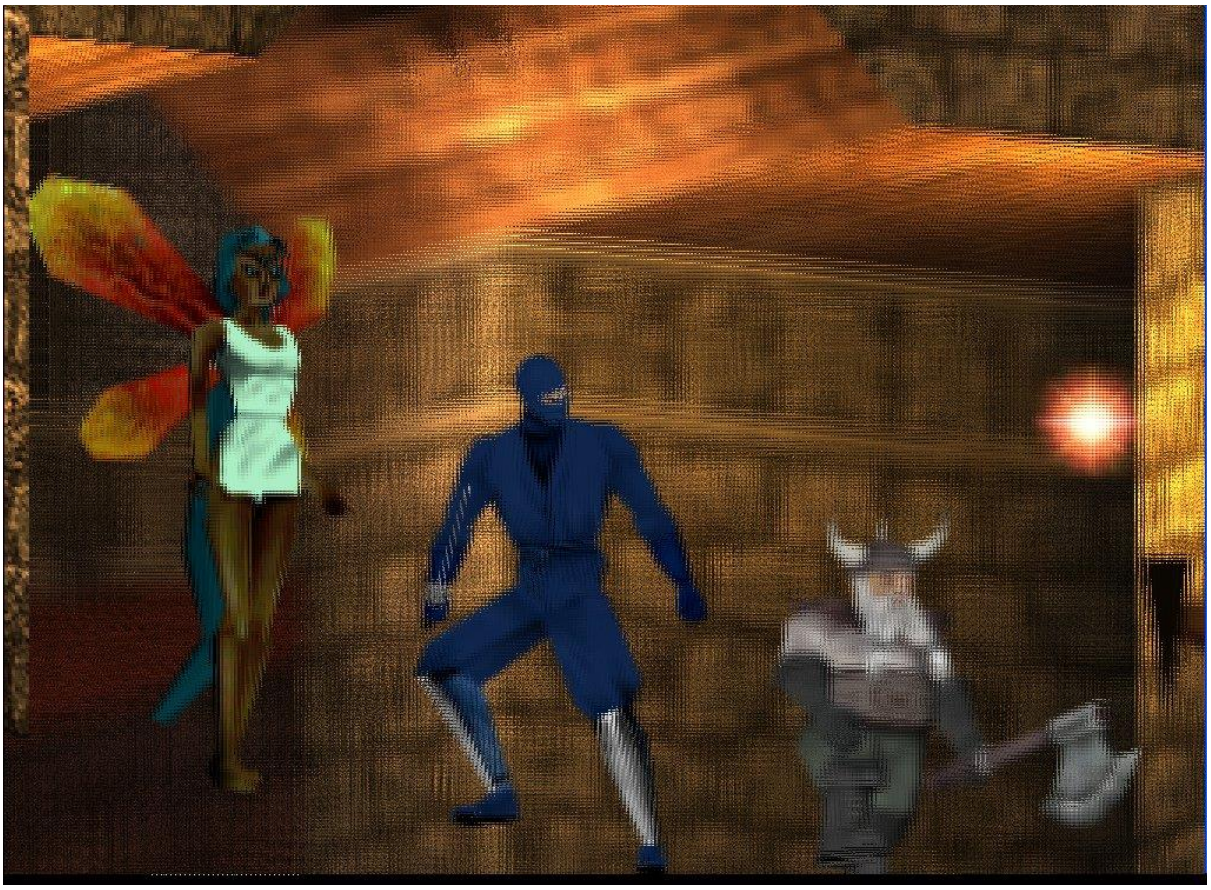


Figure 4.11: 3-D integral image (spherical) rendered with *DCTarget* and Irrlicht engine.

Figure 4.12 represents samples of *800-frame* animation converted to an animation of integral images produced based on *DCTarget* method. The integral animation can be viewed using a cylindrical lens array with the same parameters selected in the rendering process. The original 2-D animation was designed by Irrlicht. The average time needed to produce integral

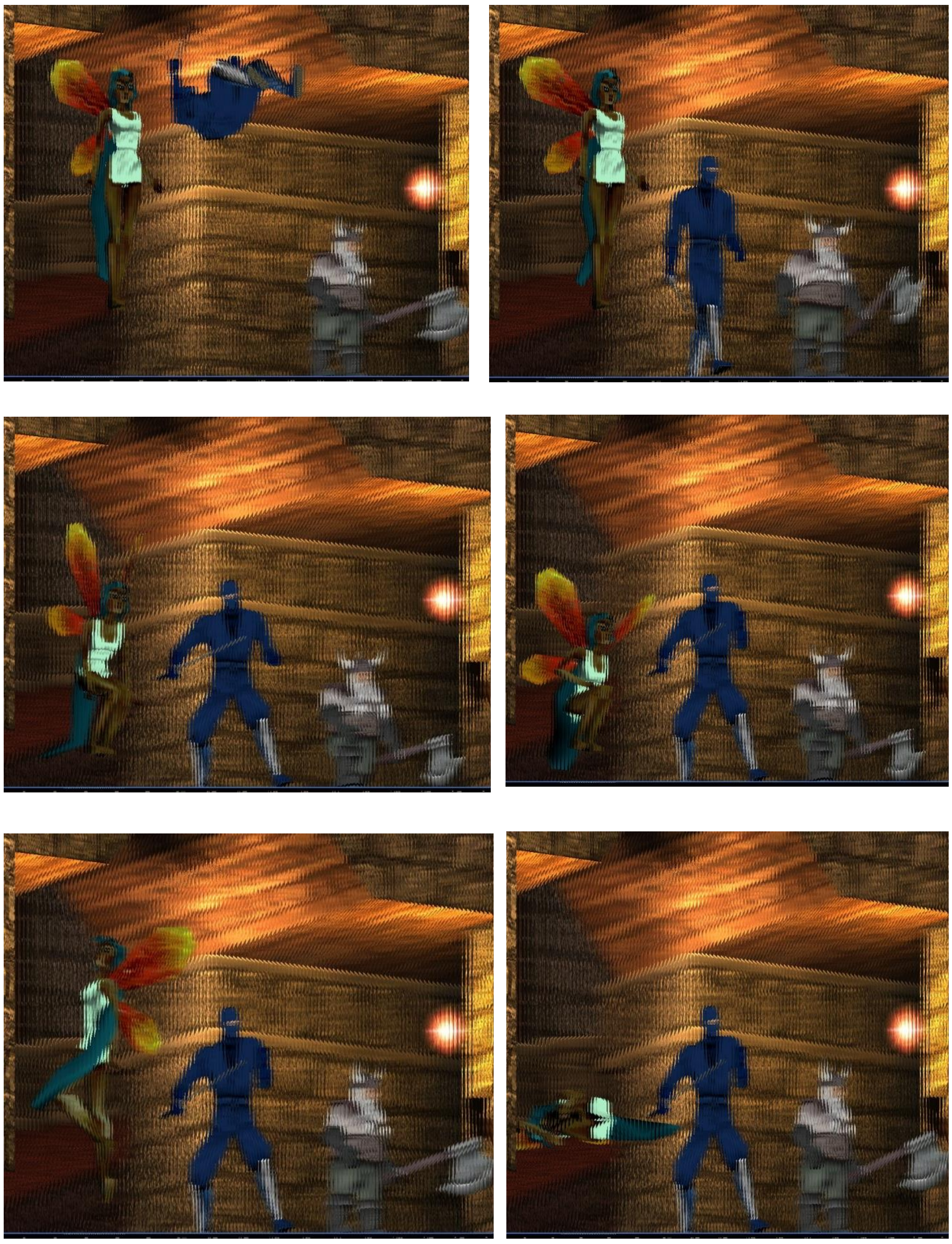


Figure 4.12: ***DCTarget*** **-based animation of integral images rendered to be viewed with a cylindrical lens array. The images are selected frames from an animation, the frame numbers are respectively 1, 150, 281, 420, 601 and 796.**

images with this method is dependent on the machine used to render the frames. In a normal PC with a *65 Mbps* video card and *3.4 GHz* processor, the average time for rendering one frame is about 4 seconds. The minimum number of frames needed for a continuous image is *24 frames per second*. The time needed to produce one second of video stream with a normal PC is about *100 seconds*

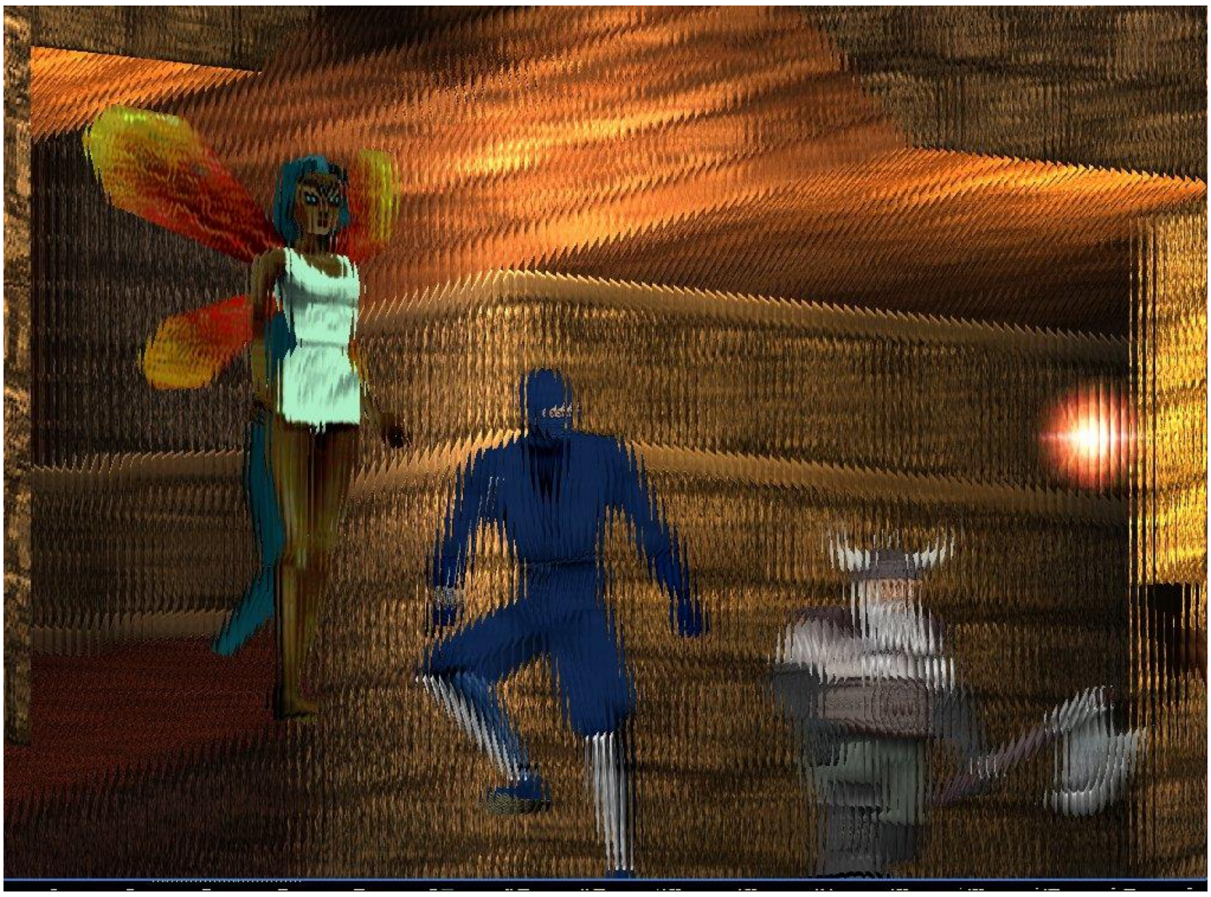


Figure 4.13: *DCTarget* -based integral image to be viewed with a cylindrical lens array.

4.9 Improvements

In order to improve the quality of the integral images produced with *DCTarget* method, and develop the implementation process of the algorithm, the following enhancements on the algorithm or the software are suggested:

4.9.1 Using two different resolutions

If a specific resolution for the capturing stage is used, and a different resolution is used for displaying the integral images, the quality of the integral images can be improved. The resolution of a printed image can be much higher than that of the available display devices.

The limitation of the available display devices resolution reduces the quality and the accuracy of the produced integral images, whereas, the resolution considered in the rendering process is the resolution of the display devices. Therefore, it would be useful if the rendering process were carried out using the resolution of the display devices, and at the same time, the static image is displayed with the resolution of the printer or the display device that has a higher resolution. For example, a cylindrical lens array is considered as a capture and display device. The number of columns under each lenslet is eight columns (*8×8 pixels* under each micro-image when a spherical microlens array is considered). The display device resolution is eight times higher than the rendering device resolution (e.g. PC screen with *1024×768 pixels* or a resolution of *75 dpi* and a printer with a resolution of *600 dpi*). Each lenslet covers *64 columns* on the printed image. The image plane should be rotated with a number of angles equals to the number of columns under each lenslet (i.e. *64 angles*).

The same algorithm is applied; the pixels are selected from the orthographic projections and mapped to the final image so that the pixel values of the images are assigned to pixels in the image plane with a higher intensity. In this case, the scene would be represented in more details and better quality, and the integral image can be stored to be displayed as a printed image. However, more complexity is added to the process. The quality of integral image is improved because more pixels are picked up from the projections, the composed image would contain much more information about the scene, and therefore, the images viewed are more continuous. Viewing the 3-D effect of the integral image happens when each eye sees columns (or pixels) extracted from different projections at the same time, thus each eye sees the scene from a different view angle. When the viewer moves, each eye sees a different set of columns or pixels extracted from different projections. The higher resolution or the higher number of projections means that the sets of columns are exchanged smoothly as if it is a real scene. Therefore, the approach of using different resolution improves the integral images quality.

One embodiment of the technique of using different resolutions is shown in Figure 4.14. In the capturing stage, some of the pixels of the orthogonal projections are selected, whereas, the reminders are discarded. Now, groups of adjacent pixels are selected instead of separated pixels or columns. An approximation is considered when selecting the pixels so that the area around the required pixel (or the column) is picked up instead of the pixel alone (or the column). In this case, no pixels are discarded and the entire orthogonal image is accommodated in the final image. In the replay stage, if a virtual spherical lens array (or lenticular) is used, each lens produces a band of parallel rays starting from its focal point, and the replayed pixels are the area extracted from the orthogonal image instead of the selected pixel (or column). The selected pixels are not necessarily accurate because the approach of selecting the pixels is approximated. However, the resolution considered for the orthographic projections is limited; therefore, if the image plane is rotated with angles corresponding to the pixels of the final image plane which has a much higher resolution, the selected pixels would be slightly different to the group of pixels that is selected with this approximated approach. Figure 4.14, right, the area (*A*) in the orthographic image is accommodated in the area (*a*) in the final image. In order to select the accurate pixels, the image plane should be rotated with angles around the angle θ and pick up the required pixels (or columns) from each orthographic image. With the previous approximation, area (*A*) is mapped to the area (*a*).

The resolution of the final image is *MR*, and the resolution of the sub-images is *R*. With a good approximation, the central ray of each lens is considered as a central ray for the band of rays collected and displayed by each lens. The whole number of pixels can be mapped to the final image where the pixels representing the band are accommodated. In this case, the number of pixels mapped under each lens in the final image is approximately equal to the number of pixels in the area (*A*) in the rotated plane multiplied by the number of the rotated planes.

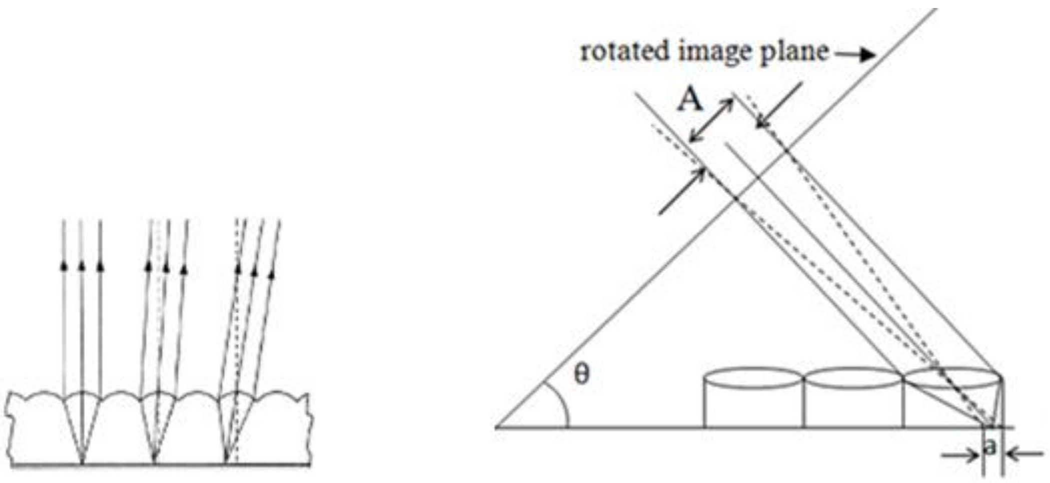


Figure 4.14: Left, replayed pixels. Right, mapped groups of pixels in the capture stage.

For example, if the resolution of the sub-images is *70 dpi*, and the final image resolution is *560 dpi*, the number of rotations is eight. With good approximation, all the pixels of each sub-image in the rotated image planes are mapped to the final image. The central ray of each group is perpendicular to the rotated image planes. In the display stage, the lenses reproduce the original rays of each rotated image and the produced rays are integrated to form the objects of the scene with lower accuracy due to the approximation we used.

If a horizontal parallax is aimed, the number of selected horizontal pixels from the orthographic projections to be accommodated under each lens and the location of the central pixel in the group are dependent on the rotation angle of the image plane. The same is applicable when considering the vertical rotation of the image plane (See Figure 4.14).

When the horizontal rotation angle is θ, the distance from the rotation axis of the central pixel in the group A is the distance between the rotation axis and the central pixel of group a multiplied by $\cos\theta$. The number of group A pixels is the integer part of A calculated from the Equation: $A = p \times \cos\theta \times r$, where p is the lens pitch, θ is the rotation angle, and r is the horizontal resolution of the rotated image plane. The number of pixels in the vertical direction will be $B = p \times \cos\alpha \times r'$, α is the rotation angle in the vertical direction and r' is the resolution in the vertical direction. The resulting micro-image will be a cluster of rectangles of pixels with varied dimensions selected from different orthographic projections.

Another embodiment can be implemented by selecting the resolution of the sub-images in the vertical and horizontal directions to be equal to the number of micro lenses in the vertical and the horizontal directions respectively divided by $\cos\theta$ where θ is the maximum angle of rotation required for rendering. In this case, the number of pixels under each lens in one direction is equal to the number of orthographic projections required to compose the final image. The pixel values picked from the selected locations in the orthographic projections are mapped to the correct pixels in the final image. An algorithm can be used to calculate the location of the pixels corresponding to the target pixels in the final image. Most of the pixels picked from the rotated images would be distributed to the micro-images in the final image but some of the pixels would be discarded.

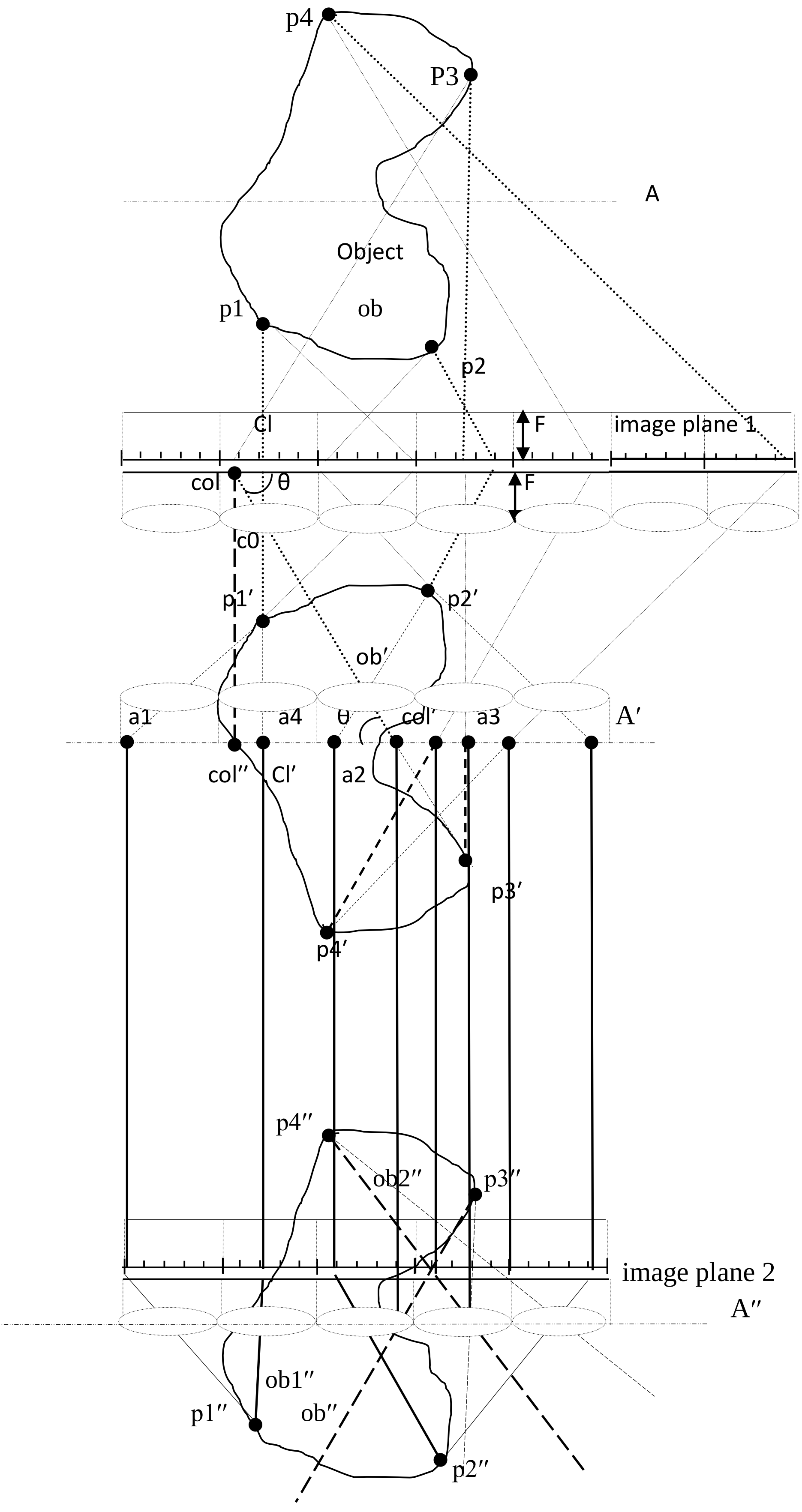


Figure 4.15: Capture, midway, and replay stages of producing integral images.

4.9.2 Display the scene in front and behind the display screen

The image composed with *DCTarget* method is the first stage of the process shown in Figure 4.15. The object points such as *p1*, *p2*, *p3*, and *p4* are projected on the image plane with *DCTarget* method through a pinhole array. The points are replayed and reconstructed with a micro lens array (or lenticular array) to form a real and pseudoscopic image to which *p1′*, *p2′*, *p3′*, and *p4′* are belonging. The following stage is aimed to correct the pseudoscopic image to be an orthogonal image. In addition, another approach can be combined with the correction process aiming to reproduce the image of the scene divided by the display screen. In this case, the viewer can view one part of the image located behind the display screen while the other part is displayed in front of the display screen.

The location of the display screen in respect to the scene can be controlled by selecting the location of Level (*A*) in the object space. Level (*A*) is an imaginary vertical plane located in the object space and parallel to the final image plane. The object points of object (*ob*) in the scene located in Level (*A*) are supposed to be the points located in the display screen when the scene is displayed. The object points of the part located in front of Level (*A*) such as *p1* and *p2* are replayed in front of the display screen. The reconstructed object points of this part such as *P1″* and *p2″* appear in front of the display screen as real and orthogonal reconstructed points, whereas, the object points behind Level (*A*) such as *p3* and *p4* are displayed as if they are located behind the display screen. The reconstructed object points such as *p3″* and *p4″* are forming an orthogonal virtual part of the scene located behind the display screen. Once the image formed on the image plane is reconstructed, Level (*A′*) contains the object points contained in Level (*A*) and the reconstructed points such as *p1′* and *p2′* are built behind Level (*A′*), while points such as *p3′* and *p4′* are reconstructed in front of Level (*A′*). The reconstructed image (*ob′*) is pseudoscopic and real image divided by the imaginary plane (*A′*).

The aim of this approach is producing a computer-generated integrated image so that the replayed image is formed of two parts; one part is a real orthogonal image in front the screen and the other is a virtual orthogonal image located behind the screen. In order to build the computer-generated image that provides the correction of the pseudoscopic image and provides the ability to display the scene in front and behind the display screen at the same time, the real image (*ob′*) is projected on an imaginary image plane located in the imaginary Level (*A′*).

The points of the reconstructed part that is going to be displayed as a real and orthogonal image located in front of the screen such as *p1′* and *p2′* are projected on plane (*A′*) to points like *a1* and *a2* respectively. The projection of each point is implemented with extending the rays that are forming the point, the points where these rays hit plane (*A′*) are the projections of the point on plane (*A′*). The rays that are forming each point in the image *ob′* start from the pixels representing the point in question on the final image plane that is composed with *DCTarget* method. These rays pass through the lenses of the microlenses array and meet in the space where the point is reconstructed because of the intersection of these rays with each other. The extensions of these rays are intersected with pixels in plane (*A′*) where the values of the original pixels that have produced the rays are assigned. The projection process is applied to all the possible projections of the image points belonging to this part of the image.

The points of the reconstructed part of the scene that is supposed to be displayed as virtual, orthogonal image behind the display screen such as *p3′* and *p4′* are projected on points on plane (*A′*) such as *a3* and *a4* respectively. The projections of a point on the image plane located on plane (*A′*) are the intersection points of the rays that are forming the point with plane (*A′*) before the image point is formed. The rays forming points like *p1′* were projected using the extensions of the rays with plane (*A′*) after forming the point in question, whereas, points like *p3′* and *p4′* are projected using the intersection of the rays forming the points with plane (*A′*) before forming these points. The rays forming a point such as *p3′* hit plane (*A′*) before intersection, and the pixels were the intersection points are located are the pixels that are supposed to hold the pixel values of the pixels representing that point. The projection process is applied to all the possible projections of the image points belonging to the part of the image that is supposed to be displayed behind the screen.

At this stage, the part of the scene that is displayed behind the screen must be projected on plane (*A′*) before the part displayed in front of the screen so that the pixels representing parts of the scene in front the screen overwrite those pixels that are representing the part behind. Leaving unassigned or empty pixels in the image plane located in plane (*A′*) must be avoided, for this reason, the part of the scene behind the screen should be fully projected, afterward, the part in front is projected, and the pixel values of appearing points automatically overwrite the pixel values of the points hidden by these points.

In Figure 4.15, image plane 1 is imaginary one while image plane 2 is the real one that is used to display the integral image. The pixels of the image formed on the imaginary image plane that is located in plane (A') are mapped to the real image plane 2 keeping the location and the values of the pixels on image plane 2 the same as the pixel values and location on the imaginary image plane. In this approach, objects such as object *ob* in the object space are computer-generated objects, the replayed images such as ob' are imaginary replayed images and the locations of their points should be calculated with the application software. The projections on the imaginary image plane (A') of the pseudoscopic image points calculated with the same software. The calculated pixel values and locations are mapped to the actual image plane 2 that is used with a micro lens array to display the integral image. On both sides of image plane 2, the integral image is displayed as required.

Once image plane 2 is covered with the lens array, the objects are reconstructed with the rays produced by the pixels and passed through the lenses. Part $ob1''$ of the reconstructed object is the part that is supposed to be displayed in front of the screen, whereas, part $ob2''$ is supposed to be displayed behind the screen. The image points of $ob1''$ are formed from the rays produced by the pixels that are holding the pixel values of the projection points already calculated. The rays produced by the pixels of part $ob1''$ converge and meet in front of the display screen and form the real and orthogonal image of this part. In the same way, the pixels representing the projection of the other part image points are forming part $ob2''$ of the image. Part $ob2''$ of the reconstructed image is virtual, orthogonal and formed with the rays produced by the pixels representing this part that was supposed to be displayed behind the display screen. The rays of this part converge and meet in virtual points seem as if they are located behind the screen. The viewer views the diverged extension of the rays forming the virtual points; therefore, they seem as if they are intersected behind the screen and forming part $ob2''$ of the image.

Figure 4.15 shows an approximated illustration of the reconstructed image $ob2''$. The display screen is supposed to display the integral image registered on image plane 2. Practically, the thickness of the micro lens array that is employed (or lenticular array) is supposed to be very small in comparison with the dimensions of the image so that the thickness is close to zero.

The actual location of the display screen plane in respect to the reconstructed image is considered to be in the location of plane *A′′*. The drawing is not fully accurate because of the exaggerated thickness of the lens array in the diagram. With this assumption, the rays reconstructing the part *ob1′′* in front the screen can be shown in the drawing. The rays are produced from the pixels on plane *A′′* and intersected to form part *ob1′′*. However, the rays forming part *ob2′′* are not depicted correctly in the diagram because the thickness of the lens array is relatively large in comparison with the reconstructed image.
Figure 4.15 shows that each one of the pixels representing the object on plane A′ can be mapped from image plane 1 with simple calculations regardless the part of the object it represents. In other words, every pixel from image plane 1 is mapped using the same formula. The following discussion is an attempt to calculate the location of the pixels when mapping them from image plane 1 to plane A′.

The dimensions of the object space are normalized when rendering using OpenGL so that the scene volume is bounded by the planes: $x = +1.0$, $x = -1.0$, $y = +1.0$, $y = -1.0$, $z = 0.0$ and $z = -1.0$. Plane $z = 0.0$ is plane of the capturing screen. The replayed image of the first stage is located between plane $z = 0.0$ of the capturing screen and $z = +1.0$. Therefore, the coordinates of the object points are normalized and bounded by the mentioned values. From now on, a length unit is considered to replace the normalized coordinates and the real length unit. The considered unit is the width and height of one pixel. Each of the normalized width of the screen and the normalized length is equal to two. The number of horizontal pixels of the screen is equal to w, whereas, the number of vertical pixels is equal to h. Then the coordinates of the object points can be measured with the number of pixels that is equivalent to the normalized values of the coordinates. For example, the point that has x, y, and z coordinates equal to $+1$, $+1$, and -1 respectively in the normalized system has x, y, and z coordinates in the new measured system equal to $w/2$, $h/2$, and -1 respectively.

The whole scene is located between the planes $z = 0.0$ and $z = -1.0$, and the image rendered in the first stage is located between $z = 0.0$ and $z = +1$. Z-coordinates were normalized in the scene space by dividing the Z-coordinates by the total depth of the scene. In the same way, Z-coordinates in the image space are normalized, then the screen width equals to 2, and the maximum depth of the image equals to 1, therefore, from now on the depth of the image space is supposed to be measured by pixels so that the maximum depth equals to $w/2$. Plane *A′* is located at a selected normalized location so that the distance from the display screen can take any value from *0.0* to *1.0* (i.e. *0.0* at the display plane and *1.0* at the end of the image space). If a cylindrical lens array is used, and the effect of the lenses is simplified so that the horizontal ray that traverses the lens behaves as if the lens is a pinhole. In the same way, if the lens array used is a spherical one, each lens is considered as a pinhole.

If a cylindrical lens array is used, Figure 4.15 shows the columns of pixels in image plane 1. Starting from the left hand side of the screen, each column is mapped to plane A'. Rays start from a column hits plane A' in the location where the column should be mapped. If the angle of a ray with image plane 1 is θ, the angle with A' is θ. To map the columns, we need to shift them to specific locations. The new location of the column on plane A' is calculated and derived from the original location of the column on image plane 1. In this particular discussion, a special coordinate system is considered in which the upper left corner of the screen is the centre of the coordinate system, the pixel coordinates increase horizontally from left to right, whereas, these increase vertically downward. The number of horizontal pixels under a lens is n, the width of a pixel is p, and the coordinate of the centre of a pixel is the coordinate of the pixel. The coordinate of a column is x, where $x = 0.5p$ is the coordinate of the first column from the left hand side of image plane 1, $x = 1.5p$ is the coordinate of the second column and so on. The point $x0$ has x coordinate = 0. For any column from image plane 1, the coordinate x for the column of the order m is:

$$x0.x = 0.5 \times p \times (2 \times m + 1) \qquad (4.61)$$

$$\tan\theta = F/x1 \qquad (4.62)$$

$x1$ is the distance between the centre of the lens and the location of the column in question. The order of the lens starting from *0* at left is:

$l = int(m/n)$, the integer part of (m/n) (4.63)

The coordinate of the lens centre is Cl (e.g. $l = 0$ is the index of the first lens from left) to which the column is belonging is calculated from the order of a column m as follows:

$$x0.Cl = (2 \times l + 1) \times (n \times p/2) \qquad (4.64)$$

For (4.61), (4.63) and (4.64), the distance *x1* for the column with the order *m* is:

$$x1 = x0.Cl - x0.x = (2\times(int(m/n)) +1)(n\times p/2) - 0.5\times p\times(2\times m +1) \qquad (4.65)$$

From Equations (4.62) and (4.65):

$$\tan\theta = F/\left(2\times\left(int\left(\frac{m}{n}\right)\right)+1\right)\left(n\times\frac{p}{2}\right) - 0.5\times p\times(2\times m + 1) \qquad (4.66)$$

In the image plane *A′* the column of order *m* and coordinate *col* is mapped to *col′* where the horizontal ray of the column hits plane *A′*.*Cl′* is the coordinate of the lens in image plane 1 that is covering the column of interest. x, and y coordinates of the pixels in plane *A′* and image plane 1 are equal, from. (4.64): $x0.Cl' = x0.Cl = (2\times l +1)(n\times p/2)$

$$x0.col' = x0.Cl' + Cl'.col' = (2\times l +1)\times(n\times p/2) + Cl'.col' \qquad (4.67)$$

Cl′.col′ is the distance between the lens centre and the mapped column:

$$\tan\theta = c0.Cl'/Cl'.col' = col.col''/col''.col' \qquad (4.68)$$

The distance *col.col″* is the normalized distance between image plane 1 and plane *A′*. *col.col″* is selected by the user to define the location of plane *A′*. *col.col″* varies from *0* to *1* in the normalized dimensions of image space. When rendering with OpenGL, in the orthogonal projection, the dimensions of the objects located in the view volume are normalised. Normalization can be simply implemented by dividing *x*, *y* and *z* coordinates of the objects by the maximum *x*, *y*, and *z* coordinates in the scene respectively. In this case, the maximum *x* coordinate value would be *+1* at the far right edge of the screen and the minimum *-1* at the far left edge, whereas, *x = 0* is the coordinate at the centre of the screen. The same is applied to y-coordinate and z-coordinate, *y = +1* is the coordinate at the maximum upper edge, *y = -1* at the lower edge, and *y = 0* at the centre of the screen.

In this approach, the coordinates are normalized so that the minimum and maximum normalized *x*, *y*, and *z* coordinates in the object space are respectively *-1* and *+1*. The image plane in the capture stage has the same dimensions of the image plane in the replay stage, then, the same normalization can be applied to the space of image *ob'*. Therefore, the total width of the screen (i.e. image plane *1*) is *2*, and the total length, whereas, the maximum depth of the object *ob* space equals to *1*, and the maximum depth of the image *ob'* equals to *1*, then the width can be considered to be proportionally equal to double the maximum depth of the image space. *Za* is the proportion of plane *A'* selected location that varies between *0* and *1*, the maximum value *1* occurs when the distance to image plane 1 is equal to half the width of the image plane. If the total number of horizontal cylindrical lenses is *N*, and the number of pixels under each lenslet is *n*, the total number of horizontal pixels in the image plane is $N \times n$. The total width is equivalent to a normalized length of *2*, *p* is the width of a pixel, and then *col.col''* as a measured length is equivalent to the length of $(N \times n/2) \times Za$ pixels, or:

$$col.col'' = (p \times N \times n/2) \times Za \tag{4.69}$$

col''.col' is the distance with which the column should be shifted on plane *A'*; this distance is measured with pixels. *col'* is the column new location and its coordinate can be calculated and each column can be shifted to its new calculated location to form the final image on image plane 2. The x coordinates of the new location of each column when a cylindrical lens array is used and the new location of the pixels when using a spherical lens array can be calculated as follows:

$$x0.col' = x0.col'' + col''.\ col' \tag{4.70}$$

From Equations (4.70) and (4.61): $x0.col'' = x0.x = 0.5 \times p \times (2m + 1)$, From Equations (4.68): $col''.col' = col.col''/\tan\theta$, and From Equation (4.69): $col.col'' = (p \times N \times n/2) \times Za$, then:

$$x0.col' = 0.5 \times p \times (2m + 1) + col.col''/\tan\theta \tag{4.71}$$

$$x0.col' = 0.5 \times p \times (2m + 1) + \left(\left(p \times N \times \frac{n}{2}\right) \times Za\right) \times (\left(2 \times \left(int\ \left(\frac{m}{n}\right)\right) + 1\right)\left(n \times \frac{p}{2}\right) - 0.5 \times p \times (2 \times m + 1))/F \tag{4.72}$$

F is the focal length of the lens that is measured with the length units. *cons1* is a constant:

$$cons1 = p \times \left(N \times \frac{n}{2}\right) \times Za/F \tag{4.73}$$

Then, the new column (or pixel) location is:

$$x0.col' = p \times [0.5 \times (2 \times m + 1) + (cons1) \times \left(\left(2 \times \left(int\left(\frac{m}{n}\right)\right) + 1\right)\left(\frac{n}{2}\right) - 0.5 \times (2 \times m + 1)\right)] \tag{4.74}$$

$$x0.col' = p \times [(m + 0.5) + (cons1) \times \left(\left(\left(int\left(\frac{m}{n}\right)\right) + 0.5\right) \times n - (m + 0.5)\right)] \tag{4.75}$$

$M = m + 0.5$, *M* is a temporary defined variable, and *x_new* is the new location of the column (or mapped pixel).

$$x\_new = x0.col' = M \times p + (cons1) \times \left(\left(\left(int\left(\frac{m}{n}\right)\right) + 0.5\right) \times n \times p - M \times p\right) \tag{4.76}$$

$$x\_new = p \times [(m + 0.5) \times (1 - cons1) + \left(int\left(\frac{m}{n}\right) + 0.5\right) \times n \times cons1] \tag{4.77}$$

$$new\_pixel\_index = int(x\_new) \tag{4.78}$$

where *m* is the index of the column (or pixel) of interest. The new location of the column indexed with *m* is calculated using formula (4.77). For example, if *p = 0.25 mm*, *F = 3 mm*, *n = 8*, *N = 128*, and Za = 0.7, $cons1$ = *29.867*. Table 4.2 represents a few samples:

Column Index *m*	0	1	2	3	4
x_new (mm)	26.258	19.0417	11.825	4.608	-3.73
Location (*pixels*)	105.032	76.1668	47.3	18.432	-14.92
New Pixel Index	105	76	47	18	-14

Table 4.2: The new locations of the columns indexed with *m*.

The negative values are rejected; it means they are located outside the image plane. The new location measured with pixels indicates the order of new pixel where the pixel (or column) is mapped. If the first pixel is indexed with *0*, then, the index of the new pixel (or column) is the integer part of the calculated location measured with pixels which is stated in (4.74). Using the following simple piece of code, the new locations are calculated in and then the columns can be mapped to plane A', the total number of columns is 1024:

```
for (u32 column_ind = 0 ;
column_ind< 1024; column_ind++) {
int len_index = floor((float)column_ind/n_pixels);
float column_ind_2=(column_ind + 0.5)*(1 - cons_1) + n_pixels*cons_1*(len_index + 0.5);
int column_ind_1 = floor((float)column_ind_2);
```

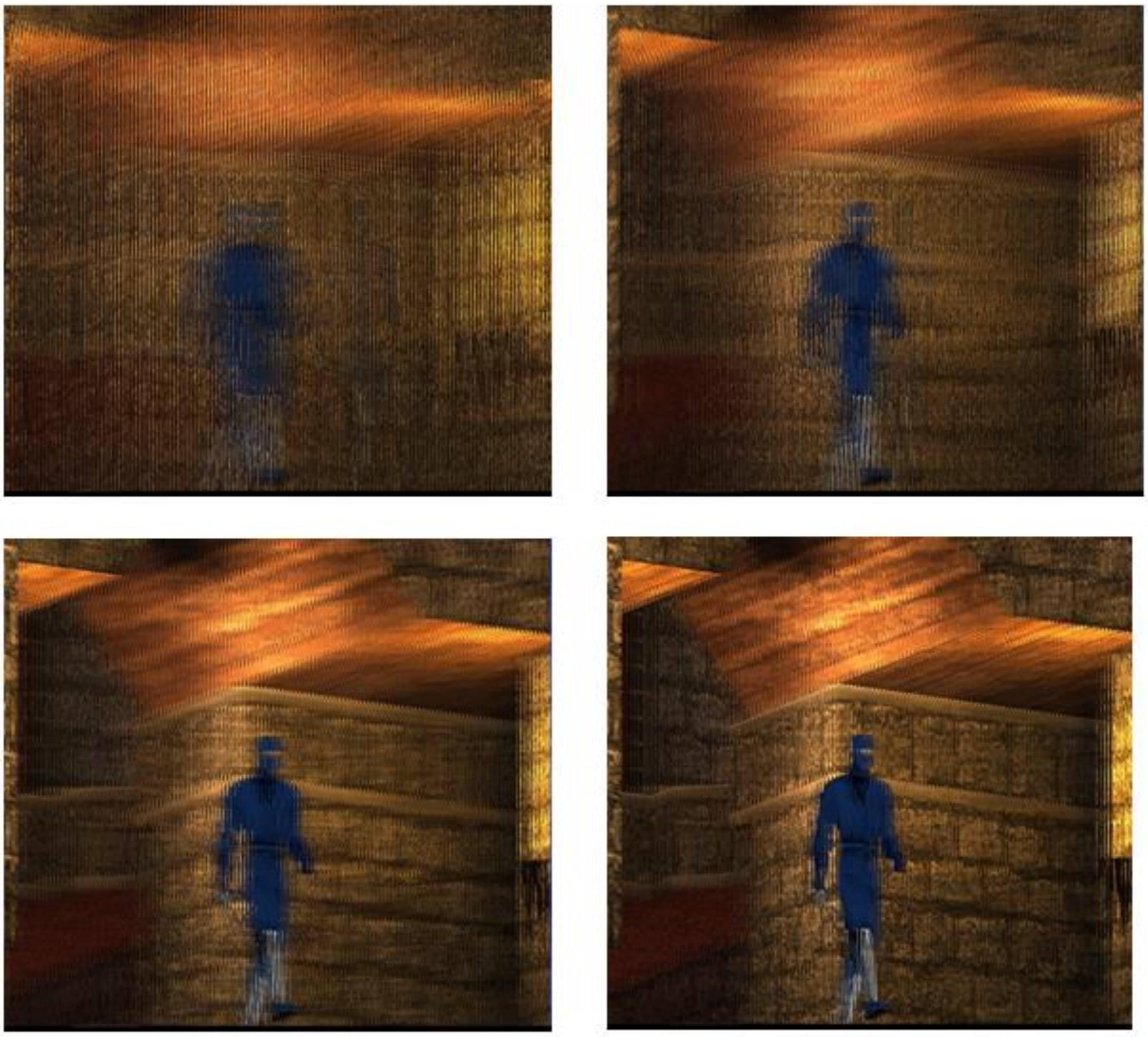


Figure 4.16: Integral images rendered for different focal length values: 5 mm (upper left), 12.5 mm (upper right), 50 mm (lower left), and 62.5 mm (lower right).

```
if(column_ind_1 >= 0)
{for (u32 row_ind = 0 ; row_ind < 768; row_ind++)
{u32 pixel_ind = column_ind*3 + 1024*row_ind*3;
image->setPixel(column_ind_1, row_ind,
video::SColor(255,pixels_index[pixel_ind],pixels_index[pixel_ind+1],pixels_index[pixel_ind
+2]), false);}}}
```

Figure 4.16 shows the resulting integral images for different focal length values when cylindrical lens array is used and the location of plane A' is equal to 0.7. The angle of view is inversely proportional to the focal length. On the other hand, when the viewing angle of the lenses is higher, each micro image is supposed to represent a larger portion of the image volume, therefore, a higher resolution of the display screen is required to display the images with a good quality.

The images are displayed with PC without natural light source. Increasing the focal length improve the quality of the image and sharpen the details of the image. However, increasing the focal length causes the view angle to be smaller and that decreases the quality. To optimize the quality, a trade-off between the view angle and the focal length should be carried out.

Integral images can be displayed with spherical micro lenses or cylindrical microlenses dependent on the way of rendering. If the image captured with virtual spherical microlens array, the image should be displayed with microlens array, the same for cylindrical lens array. The quality of spherical lens images is higher than the cylindrical lens images because the former one provides horizontal and vertical parallax, whereas, the cylindrical provides only horizontal parallax.

The quality of the resulting integral image is related to the screen resolution and the lenses focal length, pitch, and field of view. The higher number of pixels can be accommodated under each lens the better quality of image can be achieved. For a specific field of view or angle of view, pitch, and focal length, the higher resolution of the screen, the higher number for columns (when cylindrical lenses are used), or pixels (when spherical lenses are used). If more columns can be accommodated under a lenslet, a larger part of the scene can be represented with one lenslet, the micro image is sharper, and the viewer can move within the angle of view while viewing the resulting integral image with more gradual changing. If a larger part of the scene is represented with each lenslet, the lenslet produces more rays and the integration between the rays from different lenslets is more intensive, therefore, the displayed image is sharper.

In order to keep the intensity of rays produced by each lenslet when the screen resolution increases, the field of view should increases and vice versa. The screen resolution is fixed for the display image, and then the field of view or angle of view will reflect the quality of the rendered integral image. If the resolution of the display screen is limited to a specific value, we need to adjust the field of view to be small enough to allow enough pixels to be accommodated in the space on the screen that is specified for an elemental image. To solve this issue in our examples, we need to either reduce the angle or field of view, or increase the resolution. The resolution is limited so we need to reduce the field of view or the angle of view for each lenslet. The angle for each lenslet is dependent on the pitch p and thickness (i.e. the focal length) of a lenslet. For a specific resolution, a number of columns n can be held in a micro image, and the angle of view AOV is linked to the focal length f with the Equation:

$$AOV = 2\times tan^{-1}(p/(2\times f)), \text{ or, } \tan(AOV/2) = p/(2\times f) \qquad (4.79)$$

From the previous discussion and the experimental results, the image quality Q is proportional to the number of columns accommodated in a lenslet and the focal length of the lenslets, and inversely proportional to the width of the lenslet for the same number of columns. Thus, the quality can be defined with following formula:

$$Q = c\times n\times(2\times f/p) \qquad (4.80)$$

c is a constant dependent on other factors affecting the quality of the image, however, these factors are supposed to be fixed when measuring the quality in relation to the viewing angle. From Equations (4.79) and (4.80), the relation between the image quality and angle of view can be found:

$$Q =(c\times n)/(\tan(AOV/2)) \qquad (4.81)$$

AOV/2 is the maximum angle with which we need to rotate the image plane in the capture stage. From Equation (4.81), it is clear that the smaller the maximum rotation angle the image plane is rotated, the higher image quality is achieved and vice versa. In addition, the higher number of columns is accommodated under each lenslet, the better image quality can be accomplished.

The resolution considered in these examples is the resolution of the normal PC screen that is about *75 dpi*. As a result, the quality of the produced images with lower viewing angles when displayed on the normal PC screen would be higher. The focal length values selected in the examples below are *20×p = 5 mm, 50×p = 12.5 mm, 200×p = 50 mm, and 250×p = 62.5 mm, p* is the width of a pixel.

4.10 Camera model

The camera model that has been introduced in the previous chapters can be depicted in Figure 4.17. The parts called virtual image stage and image processing stage are implemented using software applications, whereas, the stage called real image stage is the actual presentation of the produced images.

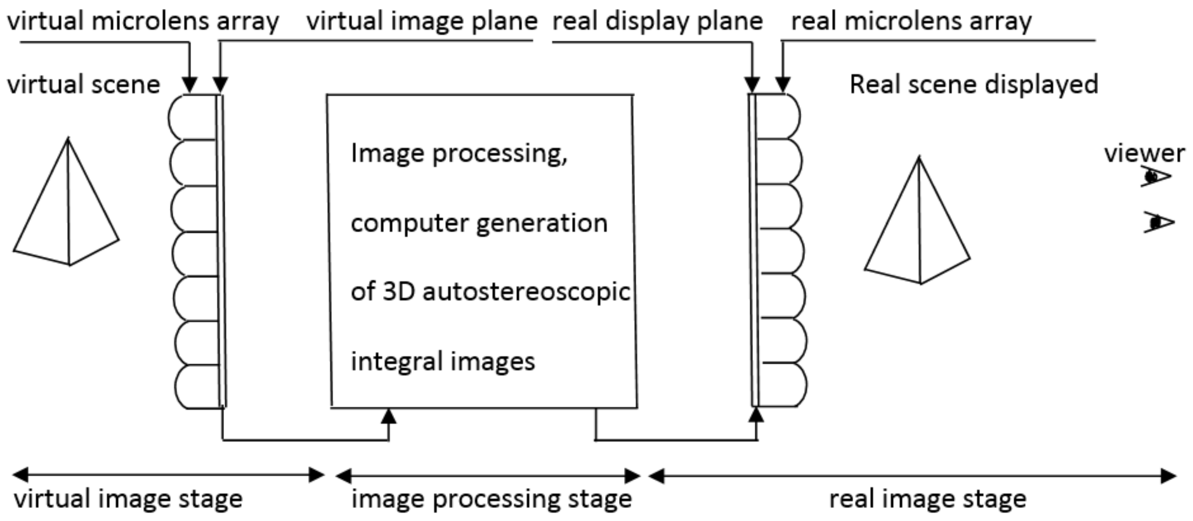


Figure 4.17: The suggested camera model

4.11 Summary

An integral image can be produced using orthographic projections of the scene on a rotating image plane that rotates with specific angles around the position of the camera while the target point is moving to specific location to provide the required rotational conditions of the image plane. The needed pixels are extracted from the resulting images and mapped to the image plane forming the integral image, whereas, the remainders of the pixels are discarded. The 3-D integral images can produce the depth effect by placing an optical device on the screen with which these images are displayed. When viewing the integral image through a cylindrical lens array or microlens array, the depth effect for each object point in the scene is linearly linked to the number of the elemental images that contain information about the projections of these points. In other words, the higher the number of elemental images that hold information about a point in the scene, the deeper that point is seen by the viewer. Therefore, in this approach, an integral image with depth effect for each point in the scene is created, or the depth information is encoded within the pixel values of the integral image and decoded by reconstructing the scene using an optical tool such as a microlens array. A virtual orthogonal scene is reconstructed and the proportional depth effect of each object point is created. The application software based on this method can be developed to render images with higher width and length and print them out to be used for applications such as advertisements.

Regarding the fact that the quality of the resulting images is dependent on the resolution of the display screen, and the resolution of the PC screens with which these images are meant to display, the quality of the resulting image is limited. However, printers provide much higher resolution, and therefore, printing the images on transparent films and displaying the images with a normal light source provides 3-D images displayed with proportionally high quality.

Chapter 5
The autostereoscopic integral images generating tools

5.1 Introduction

The application software provided with a user-friendly interface that is capable to produce autostereoscopic integral images is the main target of this chapter. In this chapter, the stages needed for producing the integral images from the start to the end are explained. The software tool utilised to implement *DCTarget* method is the plug-in tool or the application software that is intended to produce the 3-D integral images. The hardware devices are the physical tools that are employed to generate the required integral images and display the resulting images such as the PC and the lens array. The application software that is intended to design and implement should be provided with a Graphical User Interface (GUI) to allow the user to build the scene from the computer-generated models and select the measurements, the parameters, and the features of the scene. In addition, the GUI should allow the user to select the features and the parameters of the virtual camera and display devices, and then instruct the application to generate the integral images based on the *DCTarget* method with the required features and parameters.

5.2 The integral images production system structure

Figure 5.1 depicts the stages of a simple integral images production system as interacted blocks and the interactions between them. In order to produce the integral images of a scene, the scene components including the models and textures should be generated and saved in memory beforehand. The scene components should be imported to the application environment regardless the format of the files that are holding the scene geometric data or the format of the texture image files. However, specific file formats and texture file formats are supported by the implemented plug-in tool. The application environment is a Visual C++ supported by specialist libraries including Irrlicht library. In the same environment, a user-friendly GUI is used to control the scene and rendering of the integral images. Through the

GUI, the model and texture files are selected to build the static or animated scenes that are going to be converted to integral images.

The output of the rendering process would be a saved image file for a static scene, or a collection of image files for an animated scene so that each saved image file is the integral image of a single frame of the animated scene. The resulting image files can be displayed on a normal PC screen or a special display screen and viewed through a suitable lens array. The produced integral images would be viewed as a scene provided with a 3-D autostereoscopic effect. In addition, a horizontal parallax would appear when using the cylindrical lens array as a rendering mode, whereas, a horizontal and vertical parallax would appear in the case of rendering the integral images in the spherical lens array mode. The stages are going to be explained in more details.

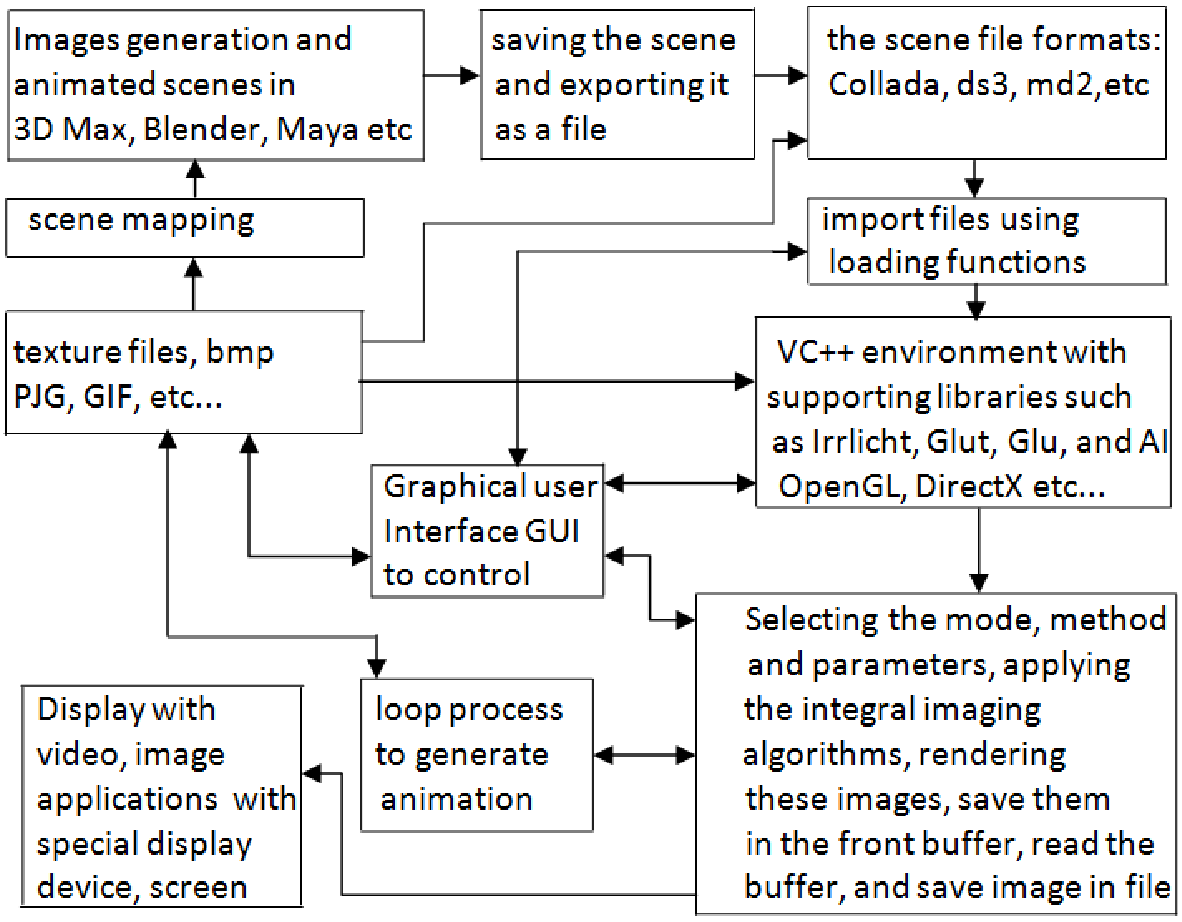

Images generation and animated scenes in 3D Max, Blender, Maya etc
saving the scene and exporting it as a file
the scene file formats: Collada, ds3, md2,etc
scene mapping
import files using loading functions
texture files, bmp PJG, GIF, etc...
VC++ environment with supporting libraries such as Irrlicht, Glut, Glu, and Al OpenGL, DirectX etc...
Graphical user Interface GUI to control
Selecting the mode, method and parameters, applying the integral imaging algorithms, rendering these images, save them in the front buffer, read the buffer, and save image in file
Display with video, image applications with special display device, screen
loop process to generate animation

Figure 5.1: The integral images production stages.

5.3 The application components and integral images production stages

5.3.1 Three-Dimension images generation

The integral images that are produced by the application are simulated integral images of the images produced by a virtual 3-D camera based on the *DCTarget* algorithm. Therefore, the images that can be converted to 3-D autostereoscopic integral images using the application software are computer-generated images. The whole scene can be generated or its components beforehand. User should be able to build the scene and add new features to the view through the GUI. The scene components can be generated using applications such as 3D Max, Blender and Maya. The whole scene can be saved in a single file, or each model can be saved apart in a separated file with one of the supported file formats. The scene can be a single static image or a film with a number of frames making an animated scene. The application should be able to import any file if its format is supported, however, importing files can imply on some difficulties because of reasons related to the program that has generated the file or even the version of the program.

Image files that hold the information needed for texture mapping should be available for rendering the models and the scene components. In addition, the texture files that are needed to build the scene through the GUI should be saved in memory and available for the program to load to the application environment. Specific image file formats are supported, however, other file format can be added and the loading functions should be developed to be able to load the newly added formats.

5.3.2 Import 3-D images

When the model file or the model frames that are forming a scene are stored in files with certain file formats, and the texture files are saved in files with certain image file formats, these files should be loaded to the environment of VC++. Selecting the file through the GUI leads to call the function employed to load the file in question. Dependent on the selected file, the correct loading function is called. Irrlicht library provides such functions that are called to load files with the supported file formats including the texture image files. In order to load model files regardless their formats, it can be useful to adopt a function that converts the files with different formats to a specific file format that can be loaded easily, and then load them to the environment.

5.3.3 Integral images production algorithm

Once the scene files including the geometric files and the texture files are loaded, the integral images production algorithm is applied to the scene to produce and display the required autostereoscopic images. The algorithm called *DCTarget* that is based on the algorithm *DIVGL* that are explained in Chapter 3 and Chapter 4 respectively are used to generate the integral images. *DIVGL* algorithm is implemented apart to produce pseudoscopic virtual integral images, whereas, *DCTarget* is used to produce orthoscopic real and virtual integral images.

5.3.4 Integral images display devices

When the integral images are rendered and saved in files, these images can be displayed on a PC screen mounted by the correct lens array to view the scene with the required 3-D effect and parallax. The higher screen resolution we use the better image quality we get. The higher resolution leads to accommodate more pixels in the same surface of the screen and therefore, the number of integrated points is higher. As a result, the image is more accurate and the continuity of the image parts is better. The thickness of normal PC screens can create an error and then affect the quality of the image because the location and value of the image pixels are calculated and rendered based on the assumption that the thickness of the screen is null. In order to get rid of this error, we need to consider the thickness of the display screen and add it to the calculations. Otherwise, the thickness of the screen should be zeroed, or reduced to the point that its effect is neglected (e.g. print the image on paper mounted by the lens array). The display screen is then mounted by the lens array. The lens array can be cylindrical if the selected rendering mode is cylindrical and spherical when the rendering mode is spherical. Several factors can affect the quality of the displayed integral image including the lens angle of view, and pureness of the lens array material.

A special display screen can be introduced; the screen should be supplied with the required cylindrical or spherical lens array as well as the LCD. On such a screen, the images can be displayed and adjusted so that the micro images of the integral images match the micro lenses of the screen lens array.

5.3.5. Graphical Unser Interface

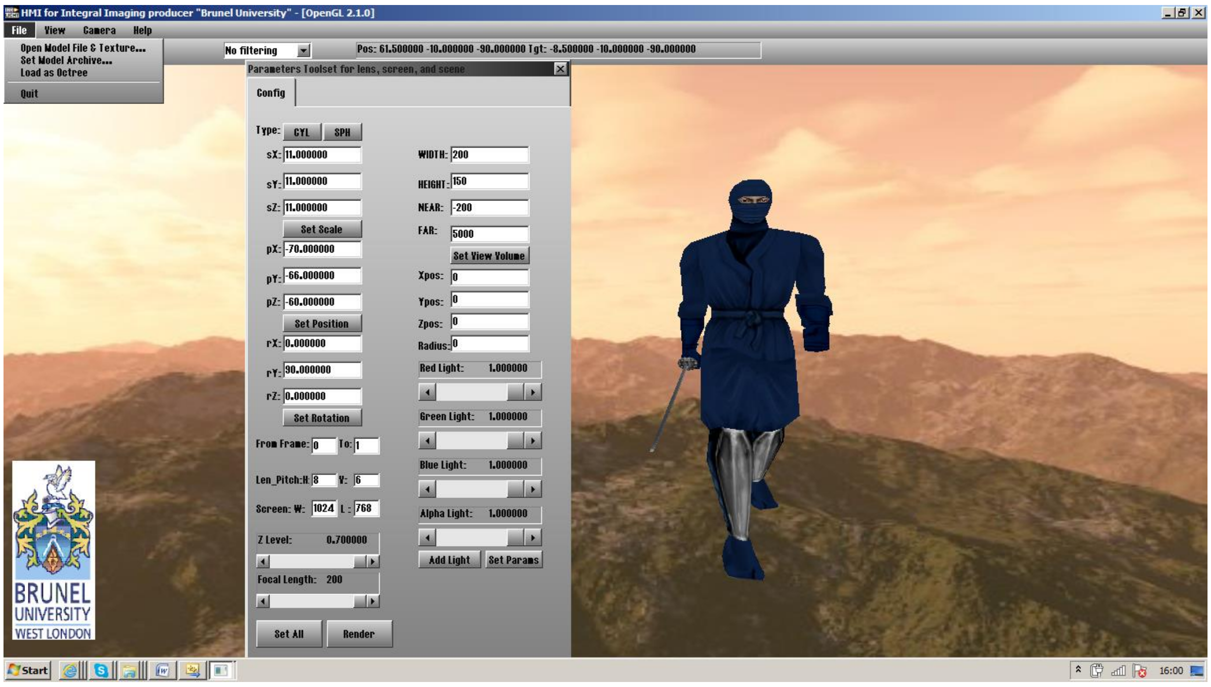
The Graphical User Interface (GUI) is based on Irrlicht engine; therefore, Irrlicht engine is referenced by including the file irrlicht.h. In addition to Irrlich and windows files, Visual C++ files needed for the GUI and the algorithms implementation are included. Name spaces such as *irr, gui, core, scene, video,* and *io* are used to include the different classes we need from Irrlich engine, windows, and VC++.

The initial setting of the program is configured by an XML configuration file. The configuration file can include the essential initial settings of the application such as the default model and texture, the welcome message etc… For example, the following simple XML file includes the name of the initial model file, and a welcome message:

```
<?xml version="1.0"?>
<config>
<!--This is a config file for Integral Imaging producer.-->
<startUpModel file="ninja.b3d" />
<messageText caption="HMI for Integral Imaging producer &quot;Brunel University&quot;">
Welcome to the HMI of the &quot;Integral Imging Producer&quot;
This program is based on Irrlicht Engine.
</messageText>
</config>
```

At the beginning, when the application starts running, a welcome message appears, and the default model is rendered with the default texture background. After closing the welcome message, the parameters toolset will be ready to set the parameters required to render the integral images. Figure 5.2 shows the screen with the toolset and the default 2-D model. The menu bar and tool bar appear on the top, and the university logo is stamped on the start-up screen but does not appear in the resulting integral images.

The GUI is supposed to allow users to build the scene from a number of model files, textures, and light sources. From the menu, the required scene file and texture can be picked up to be loaded to the scene. In order to build the scene, several models with textures can be added one by one. The parameters, texture and light sources for each model can be set with an independent GUI. The explained GUI is a simplified GUI in which the parameters of only one model can be set with the toolset. However, several models and textures can be loaded in the scene but only the parameters of only one model can be set by the user in the simplified GUI. In addition, several light sources with different coordinates, diameters and colours can be added to the scene one by one and the resulting scene can be showed instantly on the screen. GUI can be developed to allow users to load several models and set their parameters each one apart.


HMI for Integral Imaging producer "Brunel University" - [OpenGL 2.1.0]
File
View
Camera
Help
Open Model File & Texture...
Set Model Archive...
Load as Octree
Quit
Parameters Toolset for lens, screen, and scene
Config
CYL
SPH
Set Scale
Set Position
Set Rotation
Set View Volume
Add Light
Set Params
Set All
Render
Start
16:00
BRUNEL UNIVERSITY WEST LONDON

No filtering

Pos: 61.500000 -10.000000 -90.000000 Tgt: -8.500000 -10.000000 -90.000000

Type:
sX: 11.000000
sY: 11.000000
sZ: 11.000000
pX: -70.000000
pY: -66.000000
pZ: -60.000000
rX: 0.000000
rY: 90.000000
rZ: 0.000000
From Frame: 0 To: 1
Len_Pitch:H: 8 V: 6
Screen: W: 1024 L : 768
Z Level: 0.700000
Focal Length: 200

WIDTH: 200
HEIGHT: 150
NEAR: -200
FAR: 5000
Xpos: 0
Ypos: 0
Zpos: 0
Radius: 0
Red Light: 1.000000
Green Light: 1.000000
Blue Light: 1.000000
Alpha Light: 1.000000

Figure 5.2: A simple GUI to set the parameters to render Integral Images.

The GUI allows users to build the scene as well as adjusting the light, the texture, the background, the initial centre of the camera, the initial camera target, and set the parameters of the display screen, the parameters of the lenses, the position, the rotation angles, the scale of each loaded model, and the display mode. Other features and more facilities can be added to provide a more controllable rendering and produce higher quality integral images. Each feature in the simplified GUI is going to be explained representing its aim, the code behind it, and the possible improvements that can be added.

5.3.5.1. Loading the models and textures

After starting up, the screen will look as shown in Figure 5.2. In order to build the scene, the models and textures should be loaded from memory. From the menu bar, a browsing window pops up at clicking File, and then the memory can be browsed. The file that contains the geometric data of the required model can be selected from its location in the memory and loaded to the scene. In addition, the texture files needed I the scene can be loaded form their locations in the memory. In this simplified GUI, only one model can be loaded and its position, rotation angles, and scale can be adjusted. The events are applied to the GUI by the user, and then the GUI Elements send the events to the application. The application should contain an event receiver to receive these events. The event is the action taken by the user and applied to the GUI. The GUI elements are identified by enumeration values defined at the top of the C++ file. For example, the enumerator constant GUI_ID_OPEN_MODEL is the ID of the menu item that leads to open model files and textures. At clicking the button (File) in the menu bar, the list of items appears. If the item labelled with the title (open model file & texture) is selected as it is shown in Figure 5.2, the dialog box shown in Figure 5.3 appears.

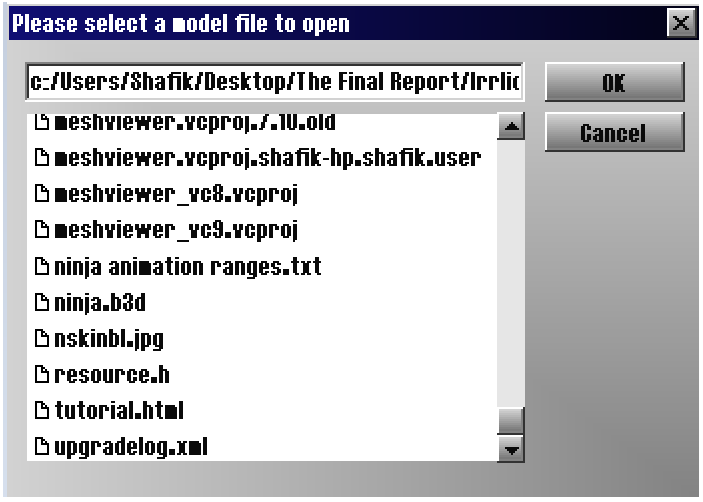

Please select a model file to open
c:/Users/Shafik/Desktop/The Final Report/Irrlic
OK
Cancel
meshviewer.vcproj.7.10.old
meshviewer.vcproj.shafik-hp.shafik.user
meshviewer_vc8.vcproj
meshviewer_vc9.vcproj
ninja animation ranges.txt
ninja.b3d
nskinbl.jpg
resource.h
tutorial.html
upgradelog.xml

Figure 5.3: An Open Model File & Texture dialog box.

Table 5.1 links each enumerator defined in the source code and its associated GUI Element:

ENUMERATOR	GUI ELEMENT
GUI_ID_X_SCALE	The value of X-coordinate of the model scale
GUI_ID_Y_SCALE	The value of Y-coordinate of the model scale

GUI_ID_Z_SCALE	The value of Z-coordinate of the model scale
GUI_ID_W_VIEWVOLUME	The width of the view volume
GUI_ID_H_VIEWVOLUME	The height of the view volume
GUI_ID_N_VIEWVOLUME	The view volume near plane z- coordinate
GUI_ID_F_VIEWVOLUME	The view volume far plane z- coordinate
GUI_ID_X_POSITION	The X-coordinate value of the model position
GUI_ID_Y_POSITION	The Y-coordinate value of the model position
GUI_ID_Z_POSITION	The Z-coordinate value of the model position
GUI_ID_X_ROTATION	The X-coordinate value of the model rotation
GUI_ID_Y_ROTATION	The Y-coordinate value of the model rotation
GUI_ID_Z_ROTATION	The Z-coordinate value of the model rotation
GUI_ID_START_FRAME	The number of the first frame of animation
GUI_ID_END_FRAME	The number of the last frame of animation
GUI_ID_HORIZONTAL_PITCH	The width of the lens pitch per pixels
GUI_ID_VERTICAL_PITCH	The length of the lens pitch per pixels
GUI_ID_SCREEN_WIDTH	The width of the display screen per pixels
GUI_ID_SCREEN_LENGTH	The length of the display screen per pixels
GUI_ID_X_LIGHTPOS	The X-coordinate of the light position
GUI_ID_Y_LIGHTPOS	The Y-coordinate of the light position
GUI_ID_Z_LIGHTPOS	The Z-coordinate of the light position
GUI_ID_R_LIGHTCOL	The light red colour scroll bar
GUI_ID_G_LIGHTCOL	The light green colour scroll bar
GUI_ID_B_LIGHTCOL	The light blue colour scroll bar
GUI_ID_A_LIGHTCOL	The light alpha scroll bar
GUI_ID_LIGHTRADIUS	The light radius value
GUI_ID_OPEN_MODEL	Open Model Files & Textures menu item
GUI_ID_SET_MODEL_ARCHIVE	Open Model archive menu item
GUI_ID_LOAD_AS_OCTREE	Open Octree Model menu item
GUI_ID_R_LIGHTCOL_VALUE	The light red colour value (from 0 to 1)
GUI_ID_G_LIGHTCOL_VALUE	The light green colour value (from 0 to 1)
GUI_ID_B_LIGHTCOL_VALUE	The light blue colour value (from 0 to 1)
GUI_ID_A_LIGHTCOL_VALUE	The light alpha value (from 0 to 1)

GUI_ID_CAMERA_MAYA	Fixed target and changeable centre camera
GUI_ID_CAMERA_FIRST_PERSON	Fixed centre and changeable target camera
GUI_ID_ABOUT	About option under Help menu item
GUI_ID_QUIT	Quit option under File menu item
GUI_ID_FOCAL_LENGTH	The lenses focal length scroll bar
GUI_ID_FOCAL_LENGTH_INFO	The lenses focal length measured per pixels
GUI_ID_BUTTON_SET_SCALE	Set the model scale button
GUI_ID_BUTTON_SET_POSITION	Set the model position button
GUI_ID_BUTTON_SET_ROTATION	Set the model rotation button
GUI_ID_BUTTON_SET_VIEWVOLUME	Set the view volume parameters button
GUI_ID_BUTTON_SET_LIGHTPARAM	Set the added light parameters button
GUI_ID_BUTTON_ADD_LIGHT	Add a new light button
GUI_ID_BUTTON_SET_ALL	Set the rest of the entered values button
GUI_ID_BUTTON_CYLINDRICAL	Select the cylindrical lens array mode button
GUI_ID_BUTTON_SPHERICAL	Select the spherical lens array mode button
GUI_ID_BUTTON_OPEN_MODEL	Open Model Files dialog box button
GUI_ID_BUTTON_SHOW_ABOUT	Open Help dialog box button on the icons bar
GUI_ID_BUTTON_SELECT_ARCHIVE	Open Archive dialog button on the icons bar
GUI_ID_Z_LEVEL	Z level scroll bar
GUI_ID_Z_LEVEL_INFO	Z level value (from 0 to 1)
GUI_ID_BUTTON_RENDER_INI	Render integral image button

Table 5.1: The enumerators and the GUI elements.

When an event occurs, GUI checks the ID of the caller and the event type, and then the related action starts. The window shown in Figure 5.3 appears as an action after the menu item that opens a dialog is selected and the GUI element GUI_ID_OPEN_MODEL sends the event.

The function that generates the open file dialog is:

env->addFileOpenDialog(L"Please select a model file to open");

addFileOpenDialog is a function form the class *IGUIEnvironment* that is accessed by the object belonging to this class and pointed to with the pointer *env*. This function returns a pointer to the open file dialog box labelled with argument text. This action is taken when the

ID of the item command is identical to GUI_ID_OPEN_MODEL. The ID is string value from the Irrlicht type *s32* returned by the function as follows:

s32 id = menu->getItemCommandId(menu->getSelectedItem());

The event happens when a menu item is clicked. The menu is defined as an object from the class *irr::gui::IGUIContextMenu* pointed to with the pointer *menu*. If any item of this menu is selected, the function *OnMenuItemSelected* that takes the menu as an argument is called:

void OnMenuItemSelected(IGUIContextMenu menu).*

Calling this function leads to getting the ID of the event, calling the function *getGUIEnvironment()* from the class *irr::IrlichtDevice* that returns the point *env* pointing to an object from the class *IGUIEnvironment* that is the GUI environment, this function provides access to the 2-D user interface environment. The ID is used in the switch statement to decide which action to be taken, for example, if the ID is identical to the constant enumerator GUI_ID_OPEN_MODEL, *addFileOpenDialog* function is called to generate the open file dialog and add it to the GUI environment.

To describe how the event receiver works, the previous example of the event is going to be explained. The event receiver is implemented with the class *MyEventReceiver* that is inherited from the class *irr::IEventReceiver*, in this class the function *OnEvent* is defined. *OnEvent* takes the event as an argument and returns true when any event occurs on the GUI. At an event, a switch statement is implemented based on the type of the event. The event types are identified with enumerators defined by Irrlicht. The following table maps some of the enumerators to the corresponding event types used in the GUI:

ENOMERATOR	**EVENT TYPE**
EET_KEY_INPUT_EVENT	Mouse or key input applied
EET_GUI_EVENT	Event occurs
EGET_MENU_ITEM_SELECTED	A menu item was clicked
EGET_FILE_SELECTED	Open model file is selected from the dialog box
EGET_SCROLL_BAR_CHANGED	Scroll bar changed and adjusted
EGET_BUTTON_CLICKED	A button on the GUI is clicked
KEY_ESCAPE	Escape key event

Table 5.2: The enumerators and the event types.

In the main function, the object *receiver* from the class *MyEventReceiver* is defined and used as an argument in the function *createDevice*, the driver of the device created is OpenGL:
MyEventReceiver receiver;
Device = createDevice(video::EDT_OPENGL, core::dimension2d<u32>(ScreenWidth, ScreenLength),16, false, false, false, &receiver);

The GUI events are checked periodically to spot any event occurrence by calling the function *MyEventReceiver*. With this function, the mouse input is checked, if the mouse left button is pressed down, the camera is subject to move and no event can be received, in this case, the function returns false, otherwise, it returns true and the GUI is able to send events. The camera centre can be changed and the scene changes instantly by pressing the mouse left button and moving the mouse, when the camera centre is adjusted as required, the button is released, and then the event occurrence is checked periodically. When an event is sent by the GUI, the ID of the element and the event type are obtained. From the open file dialog, the required model is selected and the event is sent. Based on the type of this event, the case statement leads the function to load the file selected from the list in the dialog box by calling the function *loadModel* after assigning the name of the selected file to *dialog* object, getting the file name, and converting the file name to the string format:
IGUIFileOpenDialog dialog = (IGUIFileOpenDialog*)event.GUIEvent.Caller;*
loadModel(core::stringc(dialog->getFileName()).c_str());
loadModel takes the pointer to the file name *fn* as an argument. The object *io::path filename(fn)* is the path of the file pointed to with *fn.* The extension of the file is extracted and used to determine which type of files is the selected one. If the file that was selected from the Open File Dialog box is a texture file, the extension would be one of the following set of extensions: *.jpg, .pcx, .png, .ppm, .pgm, .pbm, .psd, .tga, .bmp, .wal, .rgb, .rgba*. The function *getTexture* returns a pointer to the texture and the function *setMaterialTexture* set the texture as a material to be added to the model and rendered: *video::ITexture * texture = Device->getVideoDriver()->getTexture(filename);*
Model->setMaterialTexture(0, texture);

In the case when the file is a texture file, the path of the texture file is assigned to the path variable *download_file_name_2*. Each one of the following global path variables is meant to hold the path string of the loaded file:
io::path download_file_name_1; holds the model file path string.

io::path download_file_name_2; holds the texture file path string.
io::path download_file_name_3; holds the path string of a file saved in Irrlicht archive.
io::path download_file_name_4; holds the file path string of the Irrlicht Octree scene.
The simplified GUI allows loading a set of files, one file from each of the previous types. In order to provide the ability to load unlimited number of files from the different listed types, a global dynamic array of strings can be defined. For each one of the types mentioned above an array is defined to store the paths so that the user can select multiple files from the Open File Dialog to load, and then the application assigns the paths of the different files to the array elements of the related array. The paths are stored in the array, and later the application loops through the array elements to load the stored files during the process of rendering the Integral Images.

If the extension of the loaded file is one of the following extensions: *.pk3, .zip, .pak* and *.npk*, the loaded file is an archive file, and the file path string is assigned to *download_file_name_3*. If the loaded file is Octree, the file path string is assigned to *download_file_name_4*. Otherwise, the loaded file is a model file holding the geometric data of the model in addition to information about the texture, the animation, and other materials of the scene. The scene can be static or animated. The function *getMesh* gets the mesh of the model and returns a pointer to an object of *IAnimatedMesh*, whereas, the function *addAnimatedMeshSceneNode* returns a pointer to an object of the class *IAnimatedMeshSceneNode* that holds the data of the scene:
scene::IAnimatedMesh m = Device->getSceneManager()->getMesh(filename.c_str());*
scene::IAnimatedMeshSceneNode animModel = Device->getSceneManager()->addAnimatedMeshSceneNode(m);*

5.3.5.2. Combo boxes

GUI should allow the user to set the parameters of each loaded model apart. These parameters include scale, position, and rotation angles of the model. As shown in Figure 5.2, for each model, user can enter the values of *x, y* and *z* scale parameters and press the button labelled with (*Set Scale*) to assign the values to the scale matrix. The model vertex local coordinates are multiplied by the scale matrix so that *x* coordinate is multiplied by the first entered value in the combo box labelled with *sX* in the GUI. *y* coordinate is multiplied by *sY*, and *x* by *sZ*. When the values are entered and the Set Scale button is pressed, the values are stored in the global variables *X_ModelScale*, *Y_ModelScale*, and *Z_ModelScale* respectively. The position parameters can be entered via the GUI in the combo boxes devoted for these values. The position parameters are the coordinates *x, y,* and *z* of the model in the global coordinate system. The combo boxes to enter these values are respectively *pX*, *pY*, and *pZ*. After entering the values, the button labelled with (*Set Position*) must be pressed to assign these parameters to the global variables of he loaded model that are supposed to hold these values; these are respectively *X_ModelPosition,Y_ModelPosition*, and *Z_ModelPosition*, whereas, these values are assigned to the graphics to update the displayed model. The rotation angles are the angles the model is rotated with around the axis *x, y,* and *z* in the global coordinate system. The combo boxes to enter these values are respectively *rX*, *rY*, and *rZ*. After entering the values, the button labelled with (*Set Rotation*) must be pressed to assign these parameters to the global variables of he loaded model that are supposed to hold these values; these are respectively *X_ModelRotation,Y_ModelRotation*, and *Z_ModelRotation*. The displayed model is updated with the new entered values of the rotation angles. The scale values are assigned when the button Set Scale is pressed, in other words, in the case that the element ID is GUI_ID_BUTTON_SET_SCALE. The following code represents the process that is carried out when the button Set Scale is pressed:

```
case GUI_ID_BUTTON_SET_SCALE: {gui::IGUIElement* root =
env->getRootGUIElement(); core::vector3df scale; core::stringc s;
s = root->getElementFromId(GUI_ID_X_SCALE, true)->getText();
scale.X = (f32)atof(s.c_str()); X_ModelScale = scale.X;
s = root->getElementFromId(GUI_ID_Y_SCALE, true)->getText();
scale.Y = (f32)atof(s.c_str()); Y_ModelScale = scale.Y;
s = root->getElementFromId(GUI_ID_Z_SCALE, true)->getText();
scale.Z = (f32)atof(s.c_str()); Z_ModelScale = scale.Z;
if (Model)  Model->setScale(scale);
updateSPRInfo(Model); }break;
```

In order to get the value entered in the combo box of the scale vector's X component. The function *getRootGUIElement()* from the class *IGUIEnvironment* returns the pointer *root* to an object from the same class that holds the root of the GUI elements. The root is the total size of the GUI screen with all the elements contained in this screen through which GUI receives the interactions from user. Within the root, all the GUI elements are stored and each element is identified by its ID. The three-dimensional vector *scale* and the string *s* from the Irrlicht type *core::stringc are* defined. The function *getElementFromId* that is a member function from the class *IGUIElement* is called. *getElementFromId* returns a pointer the element whose

ID is the argument of the function, and the function *getText* returns the text combined with the element which is the text written in the combo box of the element, and then the text is assigned to the variable *s*. *s* is converted to a string using the function *.c_str(). (f32)atof* extracts the float value of the resulting string that is assigned afterward to the x component of the model scale vector *scale.X*. When the value is acquired from the GUI, the function *setScale(scale)* from the class *ISceneNode* assigns the acquired value to the model scale to be rendered with the new scale values. The same process is repeated for y and z components. For assigning the position coordinates and the rotation angles values, the functions used to assign these values are *setPosition(position)* and *setRotation(rotation)* respectively, where *position* and *rotation* are 3-dimensional vectors. In order to update the display of the model scale, position and rotation angles, the function *updateSPRInfo(Model)* is called. In a continuous loop during the time the user is setting the parameters, GUI reads the entered characters from the window, updates the model with the new entered values, and checks if the window received new characters from the keyboard.

The function *updateSPRInfo(Model)* updates the text displayed on screen. The text is the value of the parameter in its own combo box. In the case the model is not loaded, the combo box is filled with a dash (-). If the model is loaded, the combo boxes are filled at the beginning with the initial values that are already assigned to the variables, for example, the combo box of the component *x* of the model scale is filled with *X_ModelScale*. During the process of setting the parameters, the scale, position, and rotation of the model as well as the other similar parameters are continuously extracted and written to the related combo box. The function *getElementFromId* is called to get the element dialog window that is the window of interaction with GUI:

IGUIElement toolboxWnd = Device->getGUIEnvironment()->getRootGUIElement()->getElementFromId(GUI_ID_DIALOG_ROOT_WINDOW, true).*

A pointer to that element is retuned. *getElementFromId* returns a pointer to an element from the dialog window based on its ID, for example, the element x coordinate of the model scale. The function *setText(L"-")* assigns the character (-) to the element and the dash appears in the combo box specified for the element within the window element defined above:

toolboxWnd->getElementFromId(GUI_ID_X_SCALE, true)->setText(L"-").

In the case the model is loaded, the function *getScale* acquires the scale of the model as a 3-dimensional vector and then the scale vector is assigned to the vector *scale*. The x component *scale.X* is converted to an Irrlicht string with the function *core::stringw* and the converted to a string of character to be able to display in the combo box specified for the *x* component of the scale with the function *setText* as follows:

core::vector3df scale = model->getScale();

toolboxWnd-> getElementFromId(GUI_ID_X_SCALE,true)->

setText(core::stringw(scale.X).c_str());

The same method is applied to the other components of the scale as well as the position and the rotation parameters of the model.

The function *createToolBox* creates a toolbox window for each of the parameters we need to edit in the configuration stage. If there is another toolbox already created in the environment, the old toolbox is removed. The element with the ID GUI_ID_DIALOG_ROOT_WINDOW in the GUI root of the GUI environment is extracted from the root and removed. For each parameter, an edit box is created in addition the toolbox window.

The following code creates an empty configuration window with the upper left corner *x* and *y* coordinates *400* and *45* respectively, and the lower right corner point with the x and y coordinates *770* and *800* respectively. The window is titled with the text (*Parameters Toolset for lens, screen, and scene*):

IGUIEnvironment env = Device->getGUIEnvironment();*

IGUIWindow wnd = env->addWindow(core::rect<s32>(400,45,770,800),*

false, L"Parameters Toolset for lens, screen, and scene", 0,

GUI_ID_DIALOG_ROOT_WINDOW);

The function *addWindow* from the class *IGUIEnvironment* creates an empty window element; the first parameter *core::rect<s32>(x1, y1, x2, y2)* specifies the borders of the window. The function *core::rect* is a member of the namespace *core*, its parameters are from the type *s32*, and the parameters *x1* and *y1* are the coordinates of the upper left corner of the configuration window. *x2* and *y2* are the coordinates of the lower right corner of the configuration window. The coordinates of the points inside the window element are measured in pixels. The centre of the coordinate system is the upper left corner of the screen, the *x-axis* contains the upper border of the configuration window and it is directed to the right hand side, the *y-axis* contains the left border of the window and it is directed to the lower side of the window. The second parameters defines if the dialog created in a model, if it is true that means that all other *gui* elements which were created before the window cannot be used until it is removed. The third parameter is the text displayed as a title of the window. The fourth is the parent *gui* element ID of the window. The fifth parameter is the *gui* element with which the element can be identified. The function returns a pointer to the created window and returns zero of an error has occurred.

A tab control is created, added to the environment and labelled with the text (*Config*):
IGUITabControl tab=env->addTabControl(core::rect<s32>(2,20,800,800),wnd,true, true);*
IGUITab t1 = tab->addTab(L"Config");*
The function *addTabControl* that creates the tab takes the same first and second arguments of the function *addWindow*. The parent argument (i.e. the window to which the tab is belonging) can be set to *0* to place the tab control directly in the environment. The third argument specifies if the background of the tab control should be drawn. The fourth argument specifies if a flat 3d border should be drawn, this is usually not necessary unless the control is placed directly into the environment without a window as parent. The last parameter is the ID of the tab control. The function returns a pointer to the created tab control element and returns 0 if an error occurred. This pointer should not be dropped.

Edit boxes are added to the tab. The following code creates an edit box for the x-coordinate of the model scale. A static text is added to label the edit box, in this example the text is *sX*:
env->addStaticText(L"sX:", core::rect<s32>(22,48,40,66), false, false, t1);
env->addEditBox(L"1.0", core::rect<s32>(40,46,130,66), true, t1, GUI_ID_X_SCALE);
The first argument is the text that is displayed to label the edit box. The second argument of the function *addStaticText* that creates the static text is the same first argument of the function *addWindow* with which the borders of the static text are specified. The third parameter is set to true if the static text should have a 3-d border. The fourth parameter is true if the text should wrap into multiple lines. The pointer *t1* is pointing to the tab control. *t1* includes the parent ID of the element which is the window, the ID of the element, and the bolean variable that is true if the background shall be filled. The function returns a pointer to the created static text and returns 0 if an error has occurred.
addEditBox adds an edit box, the first argument is the text to be displayed; the text is the initial value of the variable (i.g. the x coordinate of the model scale). The second parameter is the rectangle specifying the borders of the edit box. The third parameter set to true if the edit box should have a 3d border. *t1* includes the parent ID of the element (i.e. the window) that

can be set to 0 to place the edit box directly in the environment, and the ID of the element. The function returns a pointer to the created edit box and returns 0 if an error has occurred.

GUI reads the values typed and entered by the user in the combo boxes specified for the values in question and then these values are assigned to the specified variables in the application. The variables entered with the same way include the following:
x, y and *z* components of the model scale vector.
x, y and *z* components of the model position vector.
x, y and *z* components of the model rotation vector.
The number of the start frame and the end frame of the animated scene.
The horizontal and the vertical dimensions of a lens measured in pixels.
The width and the height of the intended display screen measured in pixels.
The width, the length, z coordinate of the near plane, and z coordinate of the far plane of the view volume that contains the scene for the orthographic projection.
x, y and *z* components of the light position vector, and the radius of the light.
The sequenced steps of checking the GUI for new events, writing the new values in the specified combo boxes in the GUI window element, reading the entered values, and updating the parameters with the new values are implemented continuously during the configuration process of the application. The function *updateConfigInfo* is called to update the configuration information of the mentioned parameters except the x, y, and z of the model parameters (i.e. scale, the position, and the rotation angles). *updateConfigInfo* is similar to the function *updateSPRInfo*, the only different is the later function is called instantly when the parameters is changed from the GUI, whereas, the former function is called one time to update all the other parameters at once. The model parameters must be instantly updated so that the model is displayed after each modification to allow the user to judge the model and adjust it to meet the required display in the targeted integral image.

5.3.5.3. Buttons

The buttons are created in the window to receive an event from GUI and set the variables needed for rendering. As an example of the added buttons, the buttons that are used to select the type of the lenses used is considered. The button labelled with (CYL) is pressed to select the cylindrical lenses and the one labelled with (SPH) is used for spherical lenses. The enumerators GUI_ID_BUTTON_CYLINDRICAL and GUI_ID_BUTTON_SPHERICAL are the ID of the button (CYL) and (SPH) respectively. To create the two button elements, the following code is used:

IGUIEnvironment env = Device->getGUIEnvironment();*

env->addButton(core::rect<s32>(40,20,84,40), t1, GUI_ID_BUTTON_CYLINDRICAL, L"CYL");env->addButton(core::rect<s32>(86,20,130,40), t1, GUI_ID_BUTTON_SPHERICAL, L"SPH");

The function *addButton* adds a button element to the environment or the configuration window. The parameter *core::rect<s32>(40,20,84,40)* specifies the borders of the button with the coordinates of the upper left corner of the button and the lower right corner. *t1* is the parent of the *gui* element of the button. The third parameter is the ID of the element with which the *gui* element can be identified. The fourth argument is the text displayed on the button. The function returns a pointer to the created button, and returns zero when an error has occurred. When the GUI receives the event of pressing the created button, the variable *LensType* is given the value *1* that is used later when rendering the scene (i.e. if the button was pressed the lenses type is considered as cylindrical). The same method of adding a button in the GUI window is applied for the buttons described in the following table, the first column is the text that labels the button, and the second column describes its function:

THE LABEL	THE BUTTON FUNCTION
SPH	used to select the spherical type of lenses
Set Scale	assigns the entered model scale values to their variables
Set Position	assigns the entered model position values to their variables
Set Rotation	assigns the entered model rotation values to their variables
Set All	assigns the entered values of start and end frame numbers, screen width and length, and lens horizontal and vertical pitch to their variables
Set View Volume	assigns the entered parameters of the view volume to their variables
Set Params	assigns the entered parameters of the added light to their variables
Add Light	When pressed, the entered parameters are assigned to the actual light, the light combo boxes and sliders are cleared, and a new light is created
Render	Calls the suitable function for rendering the targeted integral image

Table 5.3: The GUI buttons and their functions.

5.3.5.4. Sliders

Few sliders are added to the environment to improve the performance of the GUI. As an example, the slider of Z level variable is explained. Z level variable is the variable that is used in the rendering algorithm to determine the Z-coordinate of the reference plane that divides the displayed scene into two parts; one part looks displayed behind the screen and the other one looks displayed in front of the screen. The z level value should be between *0* and *1*.

The enumerator *EGET_SCROLL_BAR_CHANGED* from the namespace *gui* takes the Boolean value (*true*) if the event on the GUI caused any one of the sliders displayed on the interface window to change its position. When the slider moves, the ID of the slider is identified. In the case when the slider is used to select z level value, the ID is GUI_ID_Z_LEVEL. The function *getPos* is called and the variable *Za* is assigned:

```
const s32 pos = ((IGUIScrollBar*)event.GUIEvent.Caller)->getPos();
Za = (float)((int)pos/255.0);
```

The function *getPos* gets the current position of the scrollbar when the event of changing the scrollbar occurs. The function returns a variable form the Irrlicht type *s32* and assigned to the constant *pos* that is converted to integer, divided by the maximum value of 255, and converted to a float variable to be assigned to *Za* that is the required z value.

Scrollbars are added to the environment to be adjusted and used to assign values to the variables required for rendering. The scrollbar of *Za* is created with the following code:

```
IGUITab* t1 = tab->addTab(L"Config");
IGUIEnvironment* env = Device->getGUIEnvironment();
env->addStaticText(L"Z Level:", core::rect<s32>(10,485,150,510), true, false, t1);
env->addStaticText(L"", core::rect<s32>(90,485,250,510), false, false, t1,
GUI_ID_Z_LEVEL_INFO);
IGUIScrollBar* scrollbar2 = env->addScrollBar(true,
core::rect<s32>(10,510,150,525), t1, GUI_ID_Z_LEVEL);
scrollbar2->setMax(255);
scrollbar2->setPos(255);
```

A pointer *env* to the environment where the scrollbar should be added is obtained from the device with the function *getGUIEnvironment* from the class *IrrlichtDevice*. The function *addStaticText* from the class *IGUIEnvironment* is called to create a text bar above the scrollbar to display the resulting value of *Za* based on the position of the scrollbar or the initial value of *Za*. The first argument of *addStaticText* specifies the text that is going to be displayed in the rectangle, the argument *core::rect<s32>(10,485,150,510)* specifies the borders of the static text. The third argument is a Boolean variable to be set to true if the static text should have a 3-D border. The fourth argument is a Boolean variable that can be set to true if the text should wrap into multiple lines. *t1* includes the parent item of the element that is the window, the ID of the element, and a Boolean variable that can be set to true if the background shall be filled, otherwise is false, it is given the default value of false. The last argument is the pointer that points to the tab control with the name *Config*. The function returns a pointer to the created static text and returns 0 if an error occurred. The text (*Z Level :)* is created by calling the function *addStaticText*, and now the same function is called to display the value of *Z level*. The first argument is a space, and the rectangle borders that should hold the value are specified with the second argument. The element ID is specified as an argument in the function, the ID *GUI_ID_Z_LEVEL_INFO* is the enumerator that determines which information is going to be displayed on the rectangle. The function *addScrollBar* from the class *IGUIEnvironment* is called to create a scrollbar for z level. Each scrollbar in the environment window is given a different pointer name. The pointer name to the z-level scrollbar is *scrollbar2* that is returned by the function *addScrollBar*, and pointing to a member of the class *IGUIScrollBar*:

IGUIScrollBar scrollbar2 = env->addScrollBar(true, core::rect<s32>(10,510,150,525), t1, GUI_ID_Z_LEVEL);*

The first argument of *addScrollBar* specifies if the scroll bar is drawn horizontal, if it is drawn vertical the value is false. The second argument specifies the borders of the scrollbar by assigning the coordinates of the upper left corner and the lower right corner of the scrollbar (*x1, x2, y1, y2*). *t1* the parent GUI element of the scrollbar (the control tab). The fourth is the ID to identify the GUI element. The function returns a pointer to the created scrollbar, or returns *0* if an error has occurred [100] [101] [102].

The function *setMax* from the class *IGUIScrollBar* sets the maximum value of the scrollbar, the maximum position of the scrollbar is corresponding to the maximum value. The function *setPos* from the class *IGUIScrollBar* sets the current position of the scrollbar.

Once the event type is identified as a scrollbar event with the element ID EGET_SCROLL_BAR_CHANGED, the scrollbar that is changed is identified with its ID, the scrollbar ID of z level is GUI_ID_Z_LEVEL. In the case statement, if the ID matches the enumerator, the following case is implemented:

```
case EGET_SCROLL_BAR_CHANGED:
if (id == GUI_ID_Z_LEVEL) {
const s32 pos = ((IGUIScrollBar*)event.GUIEvent.Caller)->getPos();
Za = (float)((int)pos/255.0); }
else if (...) {...}          break;
```

The function *getPos* form the class *IGUIScrollBar* gets the current position of the scrollbar and returns the current position *pos* as a constant string value form the Irrlicht string type *s32*. The value needed from the scrollbar is supposed to be between *0* and *1*, therefore, The string value *pos* is converted to an integer value to be normalized or dividing by the maximum and then converted to a float value and assigned to its variable *Za*. While the scrollbar is changing, the value *Za* should be displayed above the scrollbar and updated instantly. The function *updateToolBox* is called to update the information obtained through the scrollbars. In this function, a pointer to the environment *env* is returned. From the targeted environment, the GUI root element *root* is returned, and then a pointer to the dialog window *dlg* is identified by its ID and extracted from the returned root *root*. From the dialog window pointed to with *dlg*, a pointer to the element of z-level information text is identified with its ID (i.e. *GUI_ID_Z_LEVEL_INFO*), extracted, and assigned to *ZlevelInfo* as follows:

```
IGUIEnvironment* env = Device->getGUIEnvironment();
IGUIElement* root = env->getRootGUIElement();
IGUIElement* dlg = root->getElementFromId(GUI_ID_DIALOG_ROOT_WINDOW, true);
IGUIStaticText *  ZlevelInfo = (IGUIStaticText *)(
dlg->getElementFromId(GUI_ID_Z_LEVEL_INFO, true));
```

If the object pointed to with *ZlevelInfo* is not empty, the model exists, and the model type is from the type of animated mesh, the value *Za* that is acquired from the position of the scrollbar is converted to the type *stringw*, and then converted to a string. The text that displays the z-level value in the element *ZlevelInfo* is set to the string that was converted from *Za*. If the object of *ZlevelInfo* is empty, the text is set to an empty space as this code shows:

```
if (ZlevelInfo) {if ( Model && scene::ESNT_ANIMATED_MESH == Model->getType() ) {
core::stringw str(Za);
ZlevelInfo->setText(str.c_str());}
else ZlevelInfo->setText(L"");}
```

The same method of adding a scrollbar in the GUI window is applied for the scrollbars described in the following table, the first column is the text that labels the displayed value of the scrollbar, and the second column describes its function:

SCROLLBAR VALUE	THE SCROLLBAR FUNCTION
Z Level	Adjust the relative location of Z Level, value between 0 and 1
Focal Length	Adjust the focal length of the lenses, varies from 0 to 255
Red Light	Adjust the Red component of the light, it varies from 0 to 1
Green Light	Adjust the Green component of the light , it varies from 0 to 1
Blue Light	Adjust the Blue component of the light, it varies from 0 to 1
Alpha Value	Adjust the Alpha component of the light, it varies from 0 to 1

Table 5.4: The GUI scrollbars and their functions.

5.3.5.5. Cameras

The GUI allows user to select the camera target and the camera centre. The scene is imaged with the camera having the selected characteristics (i.e. the camera centre and the camera target) and rendered to produce the targeted integral images. The camera parameters are used later in the rendering process as parameters in the following function to create camera node:
scene::ICameraSceneNode camera = smgr->addCameraSceneNode(0, vector3df(X_CameraCentre,Y_CameraCentre,Z_CameraCentre), vector3df(t_newx,t_newy,t_newz)); camera->setProjectionMatrix(matr, true);*
X_CameraCentre,Y_CameraCentre, and *Z_CameraCentre* are the camera centre coordinates. *t_newx,t_newy,*and *t_newz* are the new target coordinates calculated on the basis of the selected camera target coordinates. Figure 5.2 shows the menu bar; the camera item is created and two submenus are added. The following code implements that;
menu->addItem(L"Camera", -1, true, true);
submenu = menu->getSubMenu(2);
submenu->addItem(L"Set Camera Centre", GUI_ID_CAMERA_MAYA);
submenu->addItem(L"Set Camera Target", GUI_ID_CAMERA_FIRST_PERSON);
When the first submenu is selected from the menu, the camera centre can be changed using the mouse. Clicking the left buttons and moving the mouse force the camera centre to be changed. The scene is instantly changing as the camera centre values are updated and the

scene imaged with the new camera parameters is rendered and displayed instantly. The camera can be selected from the menu; the first camera is Maya camera and the second camera is a First Person one. Maya cameras reposition themselves relative to their target, so the location is targeted. If First Person camera is selected, the camera centre coordinates are fixed while the camera target coordinates are changeable when moving the mouse and the scene is updated for the new camera target and adjusted instantly. Pointers to the cameras are defined as an array of pointers, the first member of the array points to Maya camera, and the second member points to First Person camera. The function *addCameraSceneNodeMaya* adds a Maya camera scene node, whereas, *addCameraSceneNodeFPS* adds a First Person camera scene node. Both the functions from the class *ICameraSceneNode*, *smgr* is a pointer to the scene manager. The camera centre coordinates and camera target coordinates are respectively:

(X_CameraCentre,Y_CameraCentre,Z_CameraCentre) or (X_CC, Y_CC,Z_CC) and *(X_CameraTarget,Y_CameraTarget,Z_CameraTarget) or (X_CT, Y_CT,Z_CT)*

The camera centre coordinates that are selected by moving the mouse in Maya mode are assigned to the active camera, whereas, the camera target coordinates that are selected by moving the mouse in the First Person mode are assigned to the active camera as follows:

```
scene::ICameraSceneNode* Camera[2] = {0, 0};
Camera[0] = smgr->addCameraSceneNodeMaya();
Camera[0]->setTarget(core::vector3df(X_CT, Y_CT,Z_CT));
Camera[0]->setPosition(core::vector3df(X_CC, Y_CC,Z_CC));
Camera[1] = smgr->addCameraSceneNodeFPS();
Camera[1]->setPosition(core::vector3df(X_CC, Y_CC,Z_CC));
Camera[1]->setTarget(core::vector3df(X_CT, Y_CT,Z_CT));
```

When the camera is selected form, the active camera is nominated by calling the function *setActiveCamera(Camera[i])* that sets *Camera[i]* as an active camera:

```
case GUI_ID_CAMERA_MAYA:setActiveCamera(Camera[0]);
updateCameraParameters();break;
case GUI_ID_CAMERA_FIRST_PERSON:setActiveCamera(Camera[1]);
updateCameraParameters();break;
```

The function *setActiveCamera(Camera[i])* sets *Camera[i]* as an active camera by getting the actual active camera, disabling the actual active camera, enabling *Camera[i]* as an active camera, and set *Camera[i]* to be the active camera imaging the scene:

```
scene::ICameraSceneNode * active = Device->getSceneManager()->getActiveCamera();
```

active->setInputReceiverEnabled(false);newActive->setInputReceiverEnabled(true);
Device->getSceneManager()->setActiveCamera(newActive);
At selecting the active camera, the camera parameters are updated with the new parameters entered through the GUI when the other camera was active. This can be done using the function *updateCameraParameters* and *ICameraSceneNode* from the namespace *scene*:
ICameraSceneNode cam = Device->getSceneManager()->getActiveCamera();*
cam->setPosition(core::vector3df(_CC, Y_CC,Z_CC));
cam->setTarget(core::vector3df(X_CT, Y_CT,Z_CT));
Device->getSceneManager()->setActiveCamera(cam);
The new position and target of the camera are updated with the functions *setPosition* and *setTarget* respectively, and then assigned to the active camera with *setActiveCamera*. The default camera is Maya camera. First Person camera is selected form the menu, but to switch back to Maya camera, the button ESCAPE should be used. In order to enable or disable the active to get key or mouse inputs, the function *setInputReceiverEnabled* from the class *ICameraSceneNode* is called, if this is set to true, the camera will respond to key, whereas, *isInputReceiverEnabled* checks if the input receiver of the camera is currently enabled:
camera->setInputReceiverEnabled(!camera->isInputReceiverEnabled());
Once the parameters of the active camera are selected, they are assigned to their variables such as the selected *x* position of the camera that is assigned to *X_CameraCentre*:
scene::ICameraSceneNode cam = Device->getSceneManager()->getActiveCamera();*
X_CameraCentre = (f32)(cam->getPosition().X);
The camera position and target coordinates are displayed instantly on the screen. The position is labelled with (Pos:) and the target with (Tgt:), the code employed is such the following:
core::stringw str = L"Pos: ";str.append(core::stringw(cam->getPosition().X));
str += L" ";str.append(core::stringw(cam->getPosition().Y)); str += L" ";
str.append(core::stringw(cam->getPosition().Z));str += L" Tgt: ";
str.append(core::stringw(cam->getTarget().X));str += L" ";
str.append(core::stringw(cam->getTarget().Y));str += L" ";
str.append(core::stringw(cam->getTarget().Z));postext->setText(str.c_str());

5.3.5.6. Lights

In order to add lights to the scene, a class *LightsClass* is used to hold the parameters of a light (i.e. position, colour, and radius). An array *SceneLights* of objects form the class *LightsClass* with the length *n* is defined. Multiple lights can be added to the scene, the parameters of each light are saved in an object of the array. The number of the lights is equal to the length of the array *n*. From GUI the parameters of each light can be entered. *x*, *y*, and *z* coordinates of the light position and the radius of the light are entered through the specified combo boxes in the GUI window, whereas, the colour of the light red, green, blue, and the alpha value are entered through the scrollbars specified for these variables in the GUI window. When all the parameters of a light are entered, the button labelled with (Set Params) should be pressed to assign the parameters to the object. In order to add more lights, the button labelled with (Add Light) is pressed to increment the objects index and the temporary variables are zeroed so that they can be populated with the new parameters and assign these to the new object and so on. When the objects are assigned with the entered parameters, each light (*i*) is created using its own object *SceneLights[i]* by calling the function *addLightSceneNode*:

*scene::ILightSceneNode * Light(i) = smgr->addLightSceneNode(0,*
core::vector3df(SceneLights[i].Xpos,SceneLights[i].Ypos,SceneLights[i].Zpos),
video::SColorf(SceneLights[i].Rcol,SceneLights[i].Gcol,SceneLights[i].Bcol,SceneLights[i].Alpha), SceneLights[i].Radius);

5.4. Flow-chart and sequential process

Figure 5.4 shows a diagram of the steps implemented by the plug-in tool to create the main control tools of the interface. When running the program, an event receiver is created, OpenGL driver is called, and a device is created. Several initialised variables are employed. Using function defined in the created device class, XML reader, scene manager, and video driver are created, also, models saved in archive files can be added to the scene. With the video driver class, textures can be added to the scene. XML Reader class includes the functions needed to read the XML configuration file in which the initial model is specified. The initial model is loaded, and the starting welcome message is displayed. Scene Manager Class includes the functions that are called to create cameras, adding ambient, initial skybox texture, and lights to the scene. In addition, Scene Manger includes the function *drawAll* that is called to draw the scene with all the added elements. From the created device, an environment is defined. With the functions provided in the environment class, a root element from which a dialog windows can be created as well as toolboxes with their labels. The environment includes what the user can view of the control tools. The user interacts with the

GUI using these tools. The functions provided with the environment class are called to create a menu and its submenus, tab controls, tabs, the combo boxes called Edit Boxes, toolbars, buttons, static texts, scrollbars, messages boxes and static images such as Brunel logo.

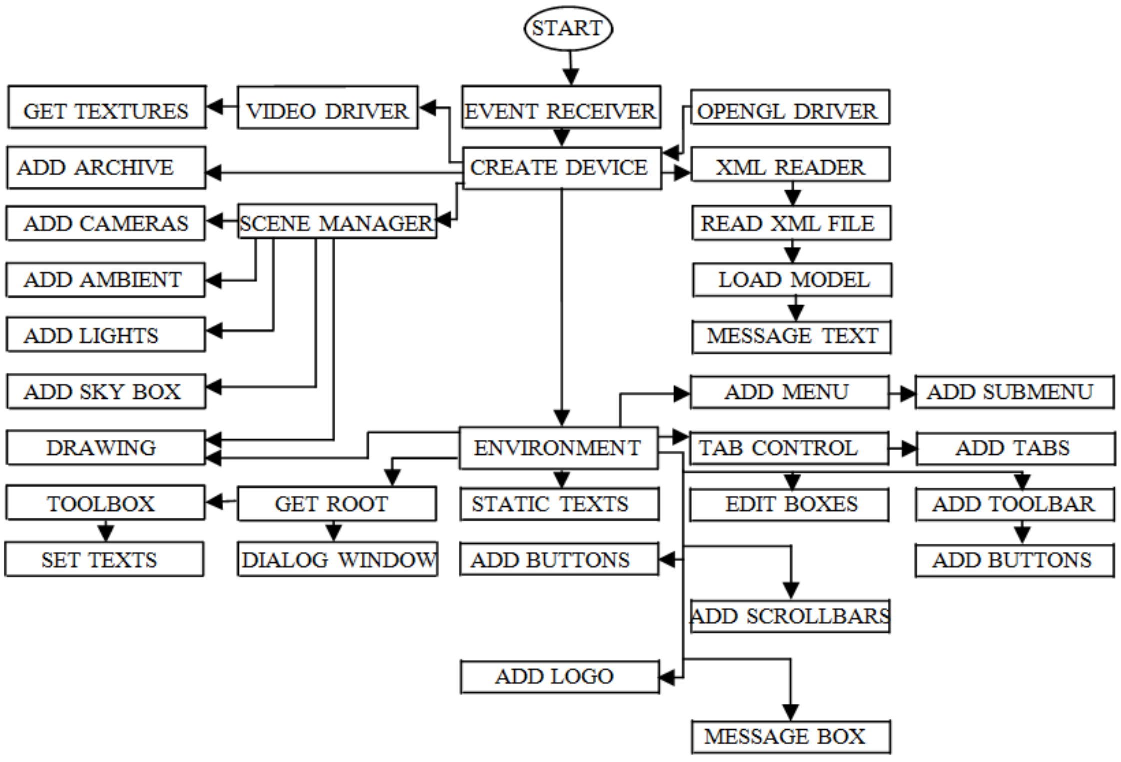


Figure 5.4: A diagram of the steps implemented to create interface control tools.

Figure 5.5, depicts a flowchart of the process implemented by the plug-in tool to produce static and animated integral images based on the DCA algorithm. Event receiver is created to receive the user actions and convert them to convenient values for using them in the program. XML configuration file can be read and the model and textures files included in the XML files are loaded initially and displayed on the screen. The environment and input control are created to provide the interface needed to enter the scene and devices parameters manually. One of the information entered manually by the user using the GUI is the lenses mode used to capture and display the scene. If the cylindrical rendering button is pressed, the cylindrical rendering algorithm explained in Chapter 4 is implemented, whereas, the spherical rendering algorithm is selected so that it is implemented when the rendering button in the GUI is pressed. At any case, each frame of the scene is rendered and converted to an integral image. The integral image for each frame is saved in a specific location in the PC memory, and then the scene is checked if it includes more frames to be rendered in sequence. The animated scenes include more than one frame. These frames are looped over to render them one by one and each frame is converted to an integral image and saved in the specified folder. When all the frames are rendered, the process is ended, and a collection of integral images is saved.

Each integral image is named with the number of the frame in the animated scene. The sequenced numbered are helpful when the animated scene is replayed by displaying the integral images in sequence.

Another manual input is using the menu items for actions such as the selection of the models and textures needed to be added to the scene. When starting up, the default models and textures specified by the XML configuration file are loaded; however, the default scene can be converted to integral images or a new scene is loaded and rendered later. To build the scene, models and textures saved in files are loaded, and other scene elements are added. From the menu bar, file button, the required models and textures can be selected and then loaded. The models and textures are saved I memory and imported to the scene environment. The loaded scene is displayed instantly on the screen. After some development, multiple models should be able to be added to the scene at the same time. Once the new scene is displayed on the screen, it is ready to be modified by the user through the GUI with the manual input.

In order to add new items to the scene, modify each model of the loaded models and set the required parameters, GUI is provided with control tools to be used. For each model, the model parameters (i.e. scale, position coordinates, and rotation angle) can be modified manually as shown in Figure 5.5, and the model with the new parameters is instantly displayed, then the user can tune these parameters to fit their requirement. In addition, the camera position and target are tuned until the required scene is displayed; the final scene acquired is the required scene to be converted to an integral image. Camera position is tuned with the mouse when Maya camera is selected from the menu. To adjust the camera target, the FP camera is selected and the target is selected with the mouse, these actions are called in the diagram as (SET MODEL PARAMS) and represented in the block named as (ENTER PARAMS). The model parameters include the first and the final frame numbers in the case of an animated scene.

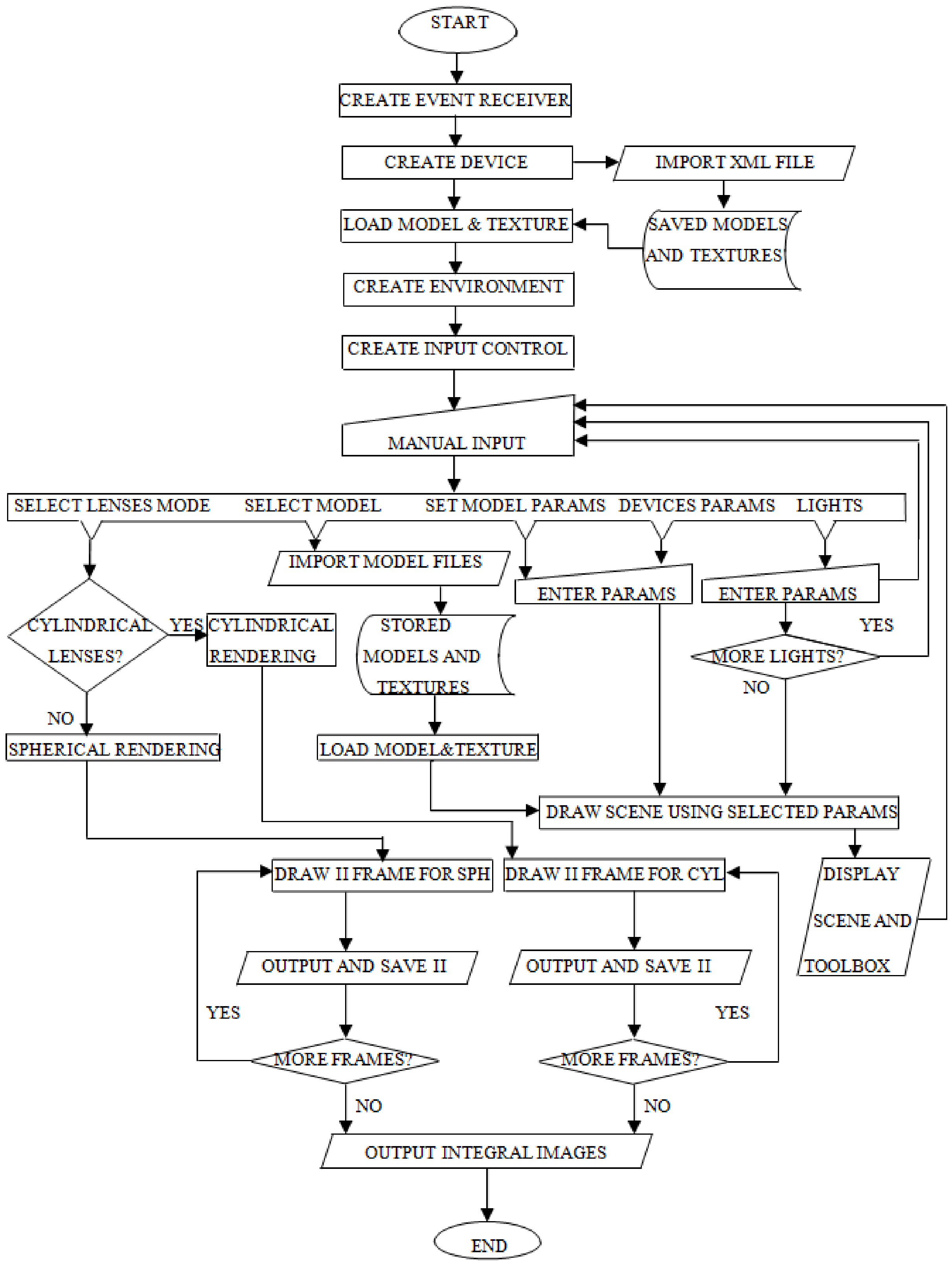


Figure 5.5: A flowchart of the integral images generation process.

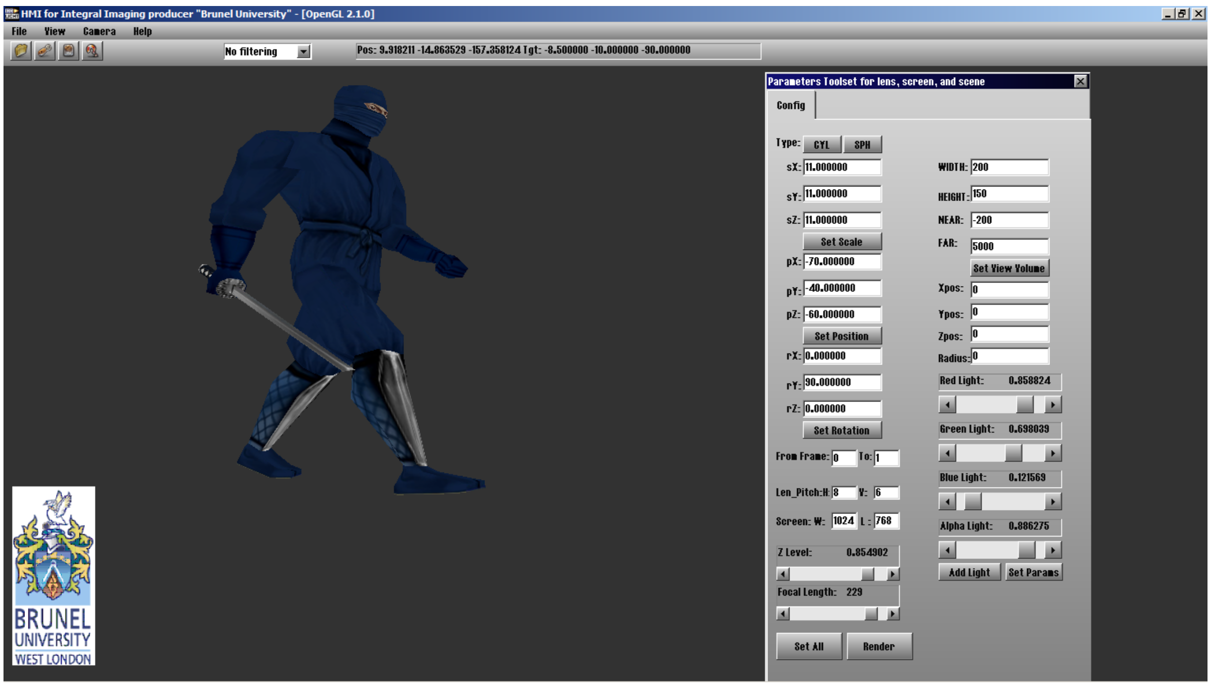
The new model parameters affect the displayed scene instantly and their values are used later in rendering the integral image. The lights that are added to the scene are represented with the action called (LIGHTS). The light parameters are entered, more lights can be added manually, and the light matrix is formed and used later in rendering. The action named (DEVICE PARAMS) is taken to enter the devices parameters manually such as the lenses, and the screen parameters. These parameters are saved and used later in rendering. Brunel University logo does not appear in the integral images.

5.5. Examples and results

Figure 5.6 shows an example when the interface window and the 2-D image that is going to be rendered as a 3-D integral image. The model and the devices parameters are selected to render an integral image of the displayed 2-D image. The lenses mode is selected to be cylindrical. The model parameters are selected to provide the required scene, and the mouse is used to set the camera parameters, while the scene is instantly displayed in a 2-D mode to allow the user to test and tune the parameters if that is needed to provide the required scene.
Figure 5.7 shows the resulting integral image of the required scene when a cylindrical lens array is used. Figure 5.8 shows another example when the interface window and the 2-D image that is going to be rendered as a 3-D integral image using a virtual spherical lens array for capturing the scene. Figure 5.9 represents the resulting integral image for the scene that was already built with the selected parameters when the spherical lens array is employed.

5.6. Set up and installation

The plug-in tool designed to implement the algorithm devoted to produce the integral images is based on the open-source Irrlicht 1-7-2 engine. The application can be run on a PC using Windows operating system. Visual C++ is the programming language used in the engine and the plug-in tool within the environment of Microsoft Visual Studio version 2008 Express Edition. OpenGL version 1.5 Application Programming Interface is used, therefore, the PC graphics card should support OpenGL version 1.5 or higher. The memory size needed to install Irrlicht engine is *170 MB*. The speed of rendering is dependent on the processor speed, for


HMI for Integral Imaging producer "Brunel University" - [OpenGL 2.1.0]
File
View
Camera
Help
Parameters Toolset for lens, screen, and scene
Config
CYL
SPH
Set Scale
Set Position
Set Rotation
Set View Volume
Add Light
Set Params
Set All
Render
BRUNEL UNIVERSITY WEST LONDON

No filtering

Pos: 9.918211 -14.863529 -157.358124 Tgt: -8.500000 -10.000000 -90.000000

Type: CYL SPH

sX: 11.000000
sY: 11.000000
sZ: 11.000000
pX: -70.000000
pY: -40.000000
pZ: -60.000000
rX: 0.000000
rY: 90.000000
rZ: 0.000000
From Frame: 0 To: 1
Len_Pitch:H: 8 V: 6
Screen: W: 1024 L : 768
Z Level: 0.854902
Focal Length: 229

WIDTH: 200
HEIGHT: 150
NEAR: -200
FAR: 5000
Xpos: 0
Ypos: 0
Zpos: 0
Radius: 0
Red Light: 0.858824
Green Light: 0.698039
Blue Light: 0.121569
Alpha Light: 0.886275

Figure 5.6: The 2-D scene for rendering with a cylindrical lens array.

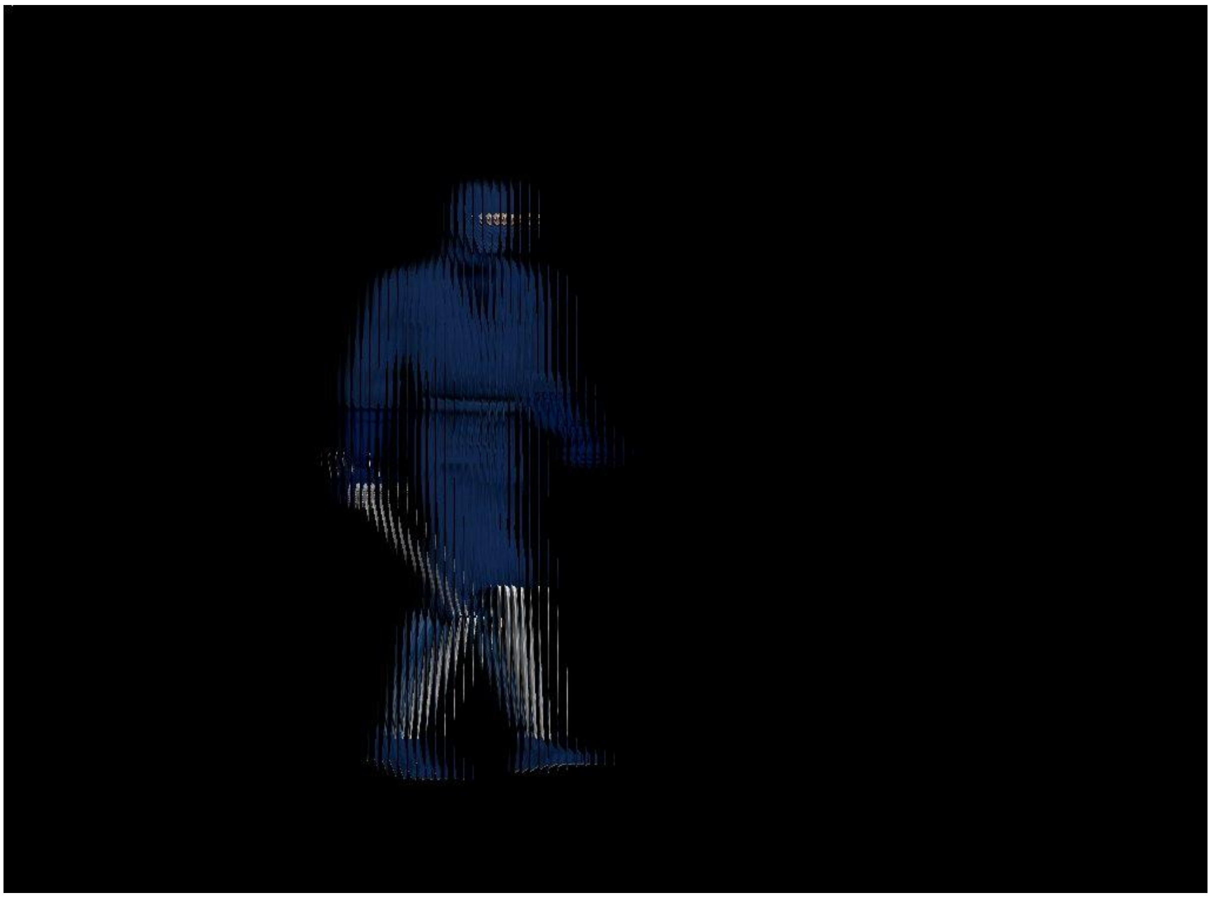
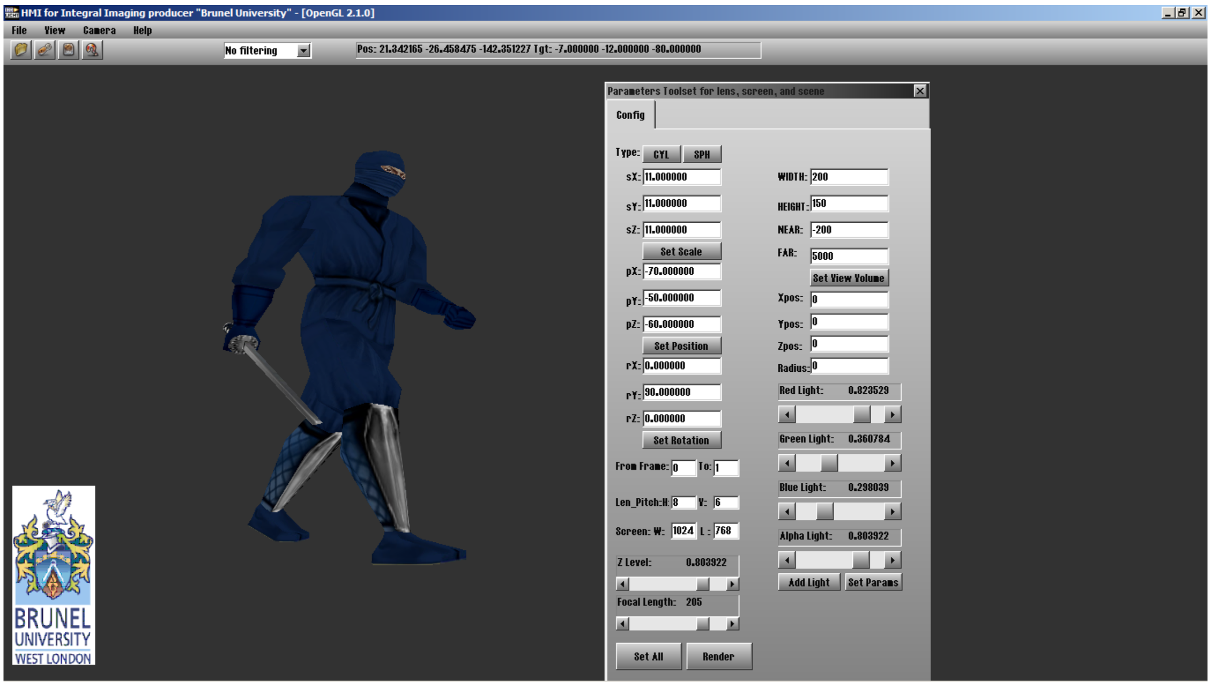


Figure 5.7: The resulting integral image based on *DCTarget* when using a cylindrical lens array and displayed on normal PC screen.


HMI for Integral Imaging producer "Brunel University" - [OpenGL 2.1.0]
File
View
Camera
Help
Parameters Toolset for lens, screen, and scene
Config
Type:
CYL
SPH
Set Scale
Set Position
Set Rotation
Set View Volume
Add Light
Set Params
Set All
Render
BRUNEL UNIVERSITY WEST LONDON

No filtering

Pos: 21.342165 -26.458475 -142.351227 Tgt: -7.000000 -12.000000 -80.000000

sX:	11.000000	WIDTH:	200
sY:	11.000000	HEIGHT:	150
sZ:	11.000000	NEAR:	-200
pX:	-70.000000	FAR:	5000
pY:	-50.000000	Xpos:	0
pZ:	-60.000000	Ypos:	0
rX:	0.000000	Zpos:	0
rY:	90.000000	Radius:	0
rZ:	0.000000	Red Light:	0.823529
From Frame: 0	To: 1	Green Light:	0.360784
Len_Pitch:H: 8	V: 6	Blue Light:	0.298039
Screen: W: 1024	L : 768	Alpha Light:	0.803922
Z Level:	0.803922		
Focal Length:	205		

Figure 5.8: The 2-D scene for rendering with a spherical lens array.

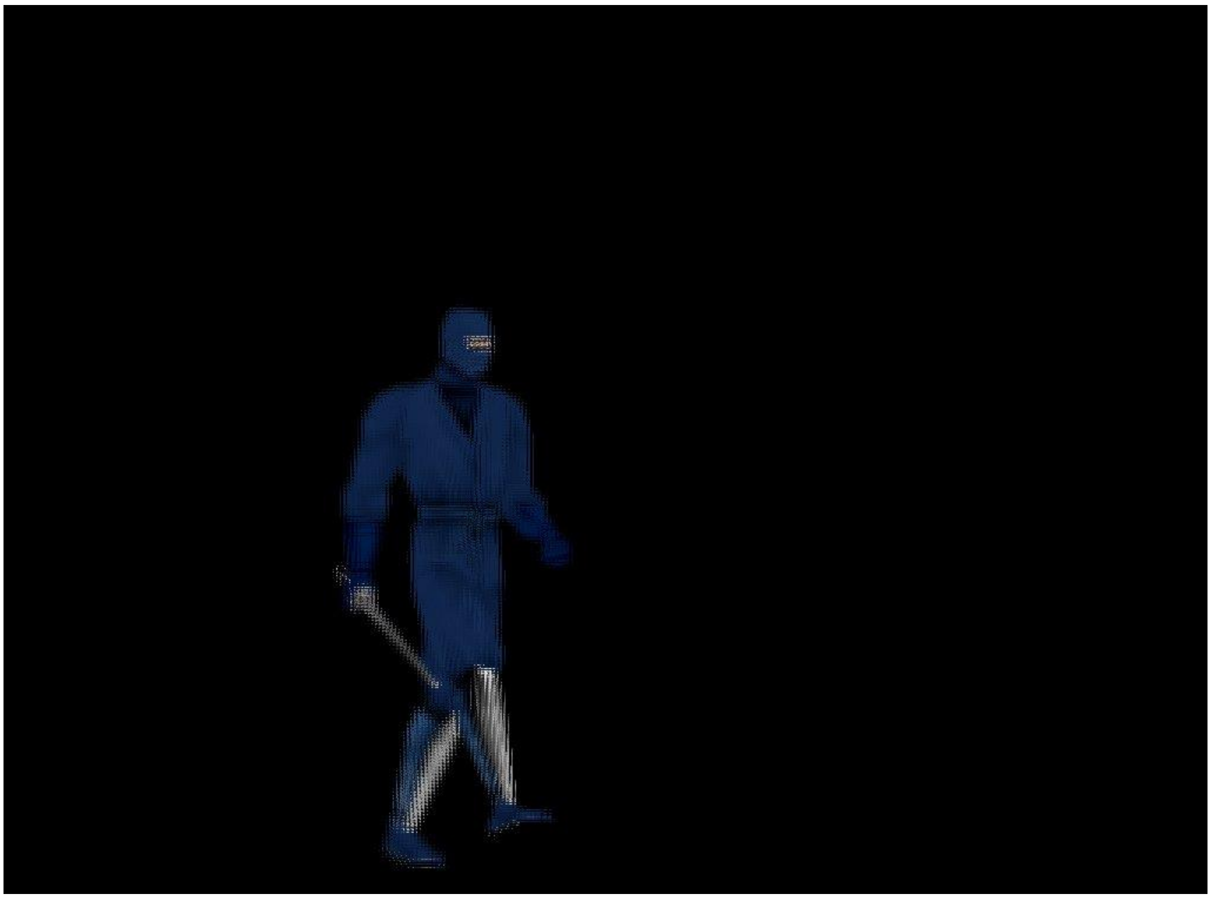


Figure 5.9: The resulting integral image based on DCTarget when using a spherical lens array and displayed on a normal PC screen.

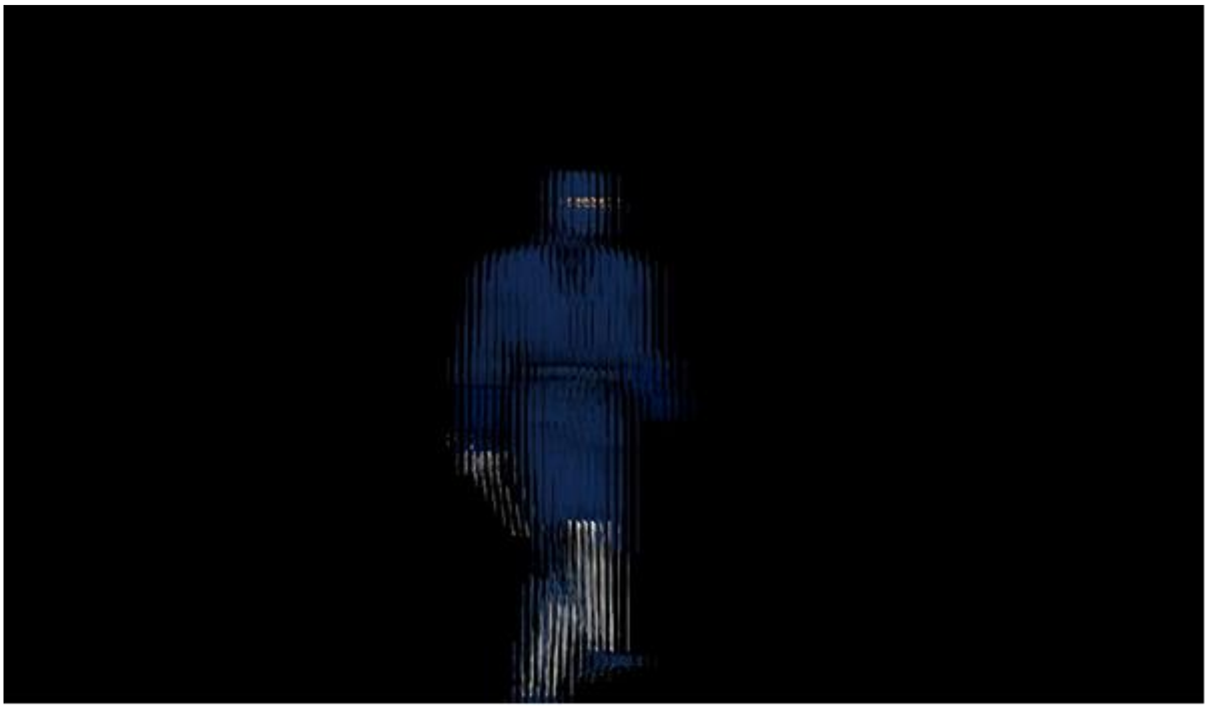


Figure 5.10: An example of an integral image of a single model.

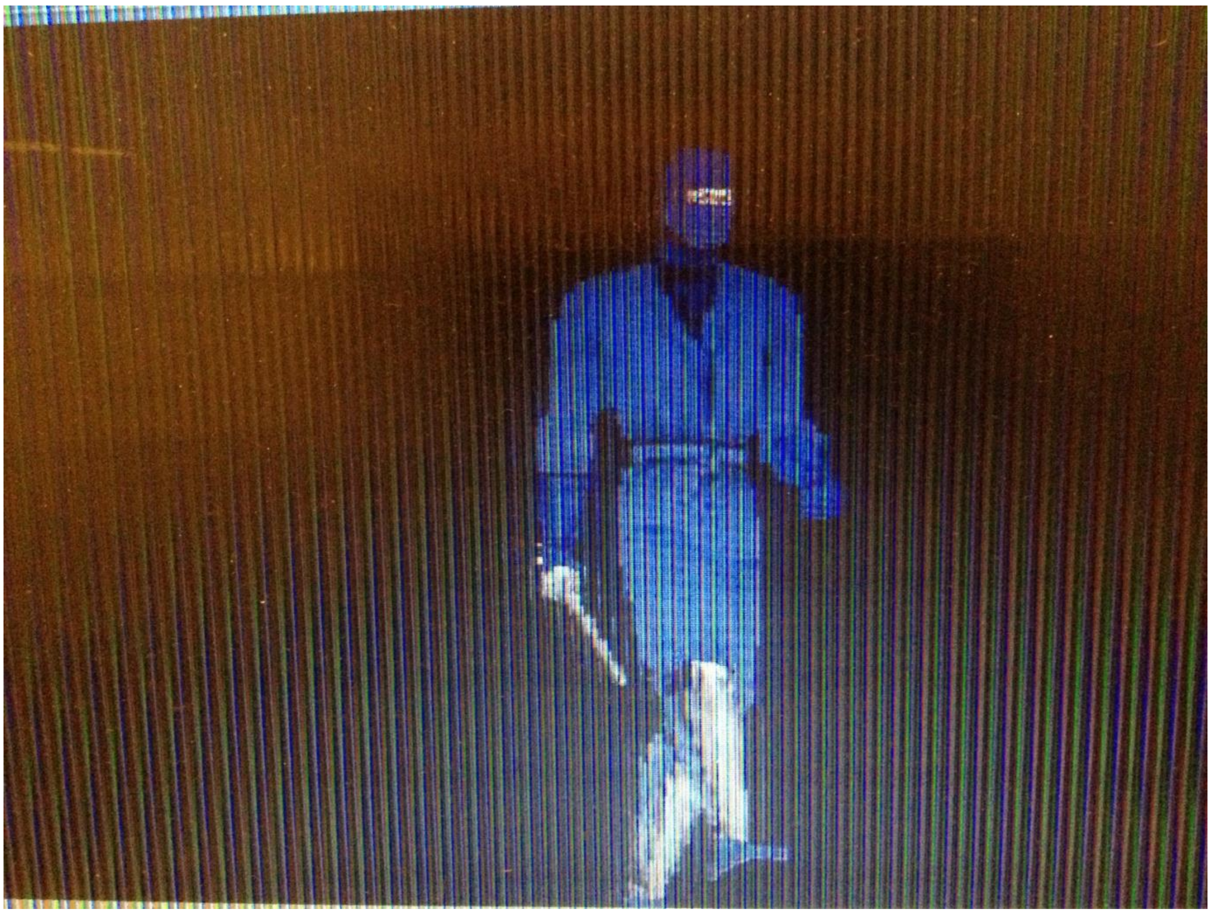


Figure 5.11: The integral image of a single model viewed with a cylindrical lens array.

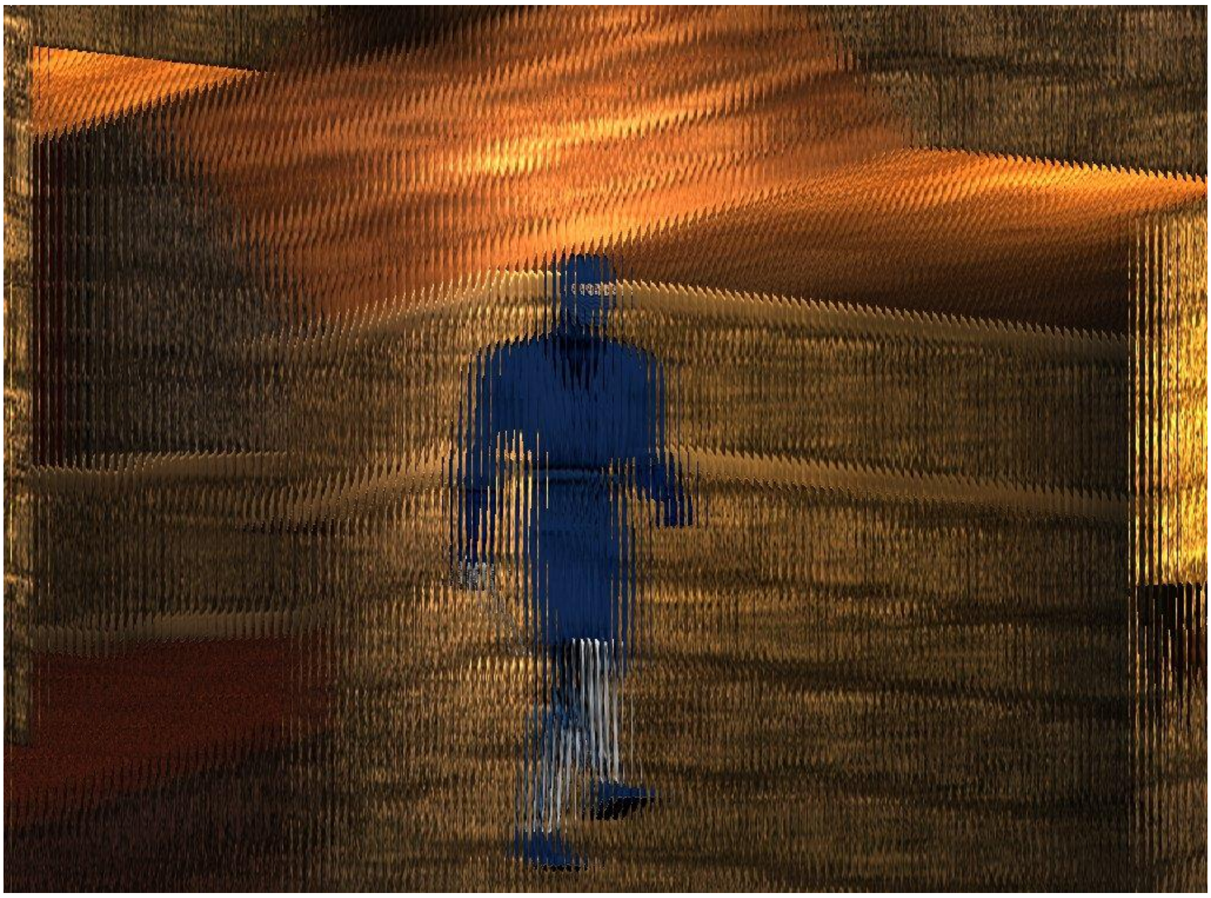


Figure 5.12: An example of a model and texture background.

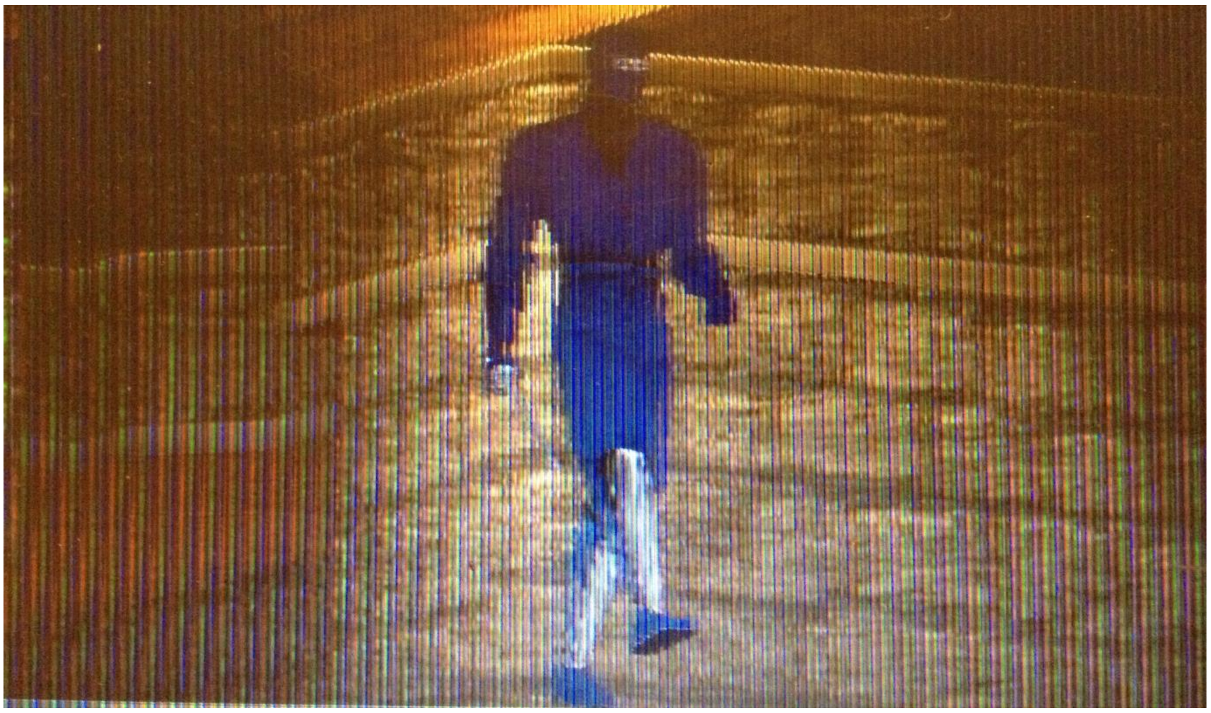


Figure 5.13: The model and texture integrated image viewed through a lens array.

example, using a PC with a processor speed of *2.1 GHz* can render each frame of an animated scene needs about 1 second to be converted to an integral image, in other words, some *60 frames* can be rendered per minute. After installing Irrlicht 1-7-2 engine, the file that contains. However, separated files can contain the functions declared or defined in the code. Other functions or classes can be contained in separated header files. Files holding the geometric or texture data of the models and the scene components should be valid and saved in memory locations so that the engine can reach them and load them to the application.

In addition, other external files such as XML file should be available to the application. After rendering the static image or the animated scene, the integral images resulting from the rendering process are saved in a specific location in the memory. The integral images can be collected and displayed on a PC screen with a suitable program such as Windows Live Movie Maker. Figure 5.10 shows an example of an integral image of a single model. The integral image was displayed on a PC screen. A primitive cylindrical lens array was used to view the image with its 3-D effect. The resulting 3-D image is shown by Figure 5.11. The same model with texture background is rendered, showed in Figure 5.12, displayed, viewed with a cylindrical lens array, and depicted by Figure 5.13. The quality of the integrated images is low because of the lack of accuracy, the low resolution of the display screen, and the low quality of the lens array.

5.7. Summary

The basic goals of the application were reached. The plug-in tool was successfully built and the algorithm of generating integral images was implemented. A simple and effective user-friendly graphical interface GUI was designed and tested. The GUI allows users to enter basic parameters of rendering easily and tune these parameters to build the required simple scenes and tailor the rendering criteria. The main problems encountered in building the plug-in tool and the application were solved such as the problem of importing models and texture files with various formats to the application. Simple scenes were built using the application and integral images were successfully rendered with a good quality. Statics images as well as animated scenes can be rendered and integral images can be produced using the plug-in tool. The resulting integral images were displayed using lens arrays and a 3D effect was produced. The application needs to be developed so that it is able to build complicated scenes, and produce higher quality images. In addition, in order to produce higher quality integral images and build complicated scenes, the GUI should allow the user to have more control over the rendering process and help the user to select from a wider range of file formats. The resulting images should be more accurate and have a better quality. In addition, the display devices should have higher resolution and better quality to be able to display such integral images correctly.

Chapter 6

Evaluation of the 3-D autostereoscopic integral images

In this chapter, objective quality assessment and complexity and speed assessment would be explained. The aim of the quality assessment is to test the quality of the images produced by the introduced two methods *DIVGL* method and *DCTarget* method, compare the two methods with each other, and compare these methods with other methods from other similar or close work. The results and analysis of the results are included in this chapter. Up to the author knowledge, the closest similar work was considered and images from that method were examined and compared against the mentioned two new methods. The closet work was achieved in De Montfort University by Graham Milnthorpe [8]. The subjective assessment was conducted in cooperation with members form CEME Company London. Complexity and speed assessment are objectively conducted by analysing the actual computational processes of the assessed methods.

6.1. SUBJECTIVE QUALITY ASSESSMENT

6.1.1. Quality of Experience

The aim of this assessment is to evaluate the two introduced methods of creating the integral images (i.e. *DIVGL* and *DCTarget*) against the closest method described in [8] and will be called Interpolative Shading Technique Method (*ISTM*). Each one of the main factors that affect the quality of a 3-D autostereoscopic integral image is evaluated apart. Factors related to the conditions, the environment, and the subjects are not considered to be contributing in the quality of the integral images generated by these two methods. Only the main factors that are relevant to the method of generating the images are considered. These factors are as follows:

1) Comfort (*a*)
2) Crosstalk, or Smooth switching from one position to another (*b*)
3) Parallax (*c*)
4) Depth of scene on both sides and view volume (*d*).
5) Field of view of integral imaging systems *FOV*, or view angle (*e*)**.**

A set of 3-D images produced with the three methods are exposed to the viewers who will be asked to evaluate each image of the group in terms of the previous stated factors. Scores should be collected from each viewer, and 15 scores will be calculated, five scores for each method, each score will be rating one of the previous five factors for each method. Mean Opinion Score (*MOS*) will be calculated for each factor of the five factors for each method, and then the total evaluation score for each method will be calculated. *MOS_x_i* for *x* test and *i* method is defined as follows:

$$MOS\_x\_i = \frac{\sum_{v=1}^{N} Rv\_x\_i}{N} \quad (6.1)$$

where, Rv_x_i is the rate that is given to the image(s) of the test *x* for the method *i* by the viewer number v, Rv_x_i can take any value from *0* to *100*. *N* is the total number of viewers. For method 1 (e.g. *DIVGL*), the *MOS* for *a, b, c, d*, and *e* factors are respectively: *MOS_a_1, MOS_b_1, MOS_c_1, MOS_d_1*, and *MOS_e_1.* For method 2 (e.g. *DCTarget*), the *MOS* for *a, b, c, d*, and *e* factors are respectively: *MOS_a_2, MOS_b_2, MOS_c_2, MOS_d_2,* and *MOS_e_2.* And so on. Providing *MOS* values are normalized by 100 (e.g.). These criteria can be looked at as dimensions and therefore, the Root Mean Square of these dimensions is proportional to the quality of the image. Based on this concept, the following formulas are introduced to calculate the Quality of Experience *QoE_1* and *QoE_2* as a percentage for method 1 and method 2 respectively:

$$A1 = (MOS\_a\_1)^2 + (MOS\_b\_1)^2 + (MOS\_c\_1)^2 + (MOS\_d\_1)^2 + (MOS\_e\_1)^2 \quad (6.2)$$

$$A2 = (MOS\_a\_2)^2 + (MOS\_b\_2)^2 + (MOS\_c\_2)^2 + (MOS\_d\_2)^2 + (MOS\_e\_2)^2 \quad (6.3)$$

$$QoE\_1 = [\sqrt{A1}/\sqrt{5}] \times 100 = [\sqrt{A1}] \times (44.72) \quad (6.4)$$

$$QoE\_2 = [\sqrt{A2}/\sqrt{5}] \times 100 = [\sqrt{A2}] \times (44.72) \quad (6.5)$$

The numerical value of the *QoE* expresses the quality of the image and therefore, the *QoE* of different methods allow comparing the quality of the images generated with the different images and as a result the quality of images the method can provide.

6.1.2. Subjective Test Conditions

Training session: after this session, the viewers will be familiarized with the produced 3-D autostereoscopic integral images and fully understanding the aim of the test and the concept of every part of the assessment. The five criteria they should focus on should be explained clearly so that the evaluation the viewer gives out is built on concrete understanding of the criteria they evaluate. The session will include the training phase and the actual test. The training phase length is dependent on the viewers who are participating in the test, if they have previous experience and knowledge about this type of images and the concepts behind the test, the training session will be shorter. The training phase time is estimated to be between 10 and 30 minutes. The estimated maximum time of the actual test phase is 83 minutes including filling in the demographic questionnaire, reading the instructions, watching the videos and the static images, scoring the images, and finally filling in the Feedback questionnaire. However, the test time can be much shorter if the viewer marked images faster.

Stimuli: 4 groups of the images and videos are selected to assess the five criteria of 3-D images that are generated with the introduced methods: Training Images, Group 1, Group 2 and Group 3. Training images that are shown in Figure 6.1 and called ***Training Globes*** are generated with a software based on the method explained in Chapter 2 (i.e. Forward Projection Pinhole Rendering Model).

- **Training Images** are used in the training session to explain the criteria the assessment is targeting and to declare how the Opinion Score should be made for each criteria. The two images are generated in a way that the left image in Figure 6.1 appears behind the display screen while the right one appears in front of the display screen. Viewer should score each image for the five criteria and these images are used in the training session to point out how the score works and on which basis.

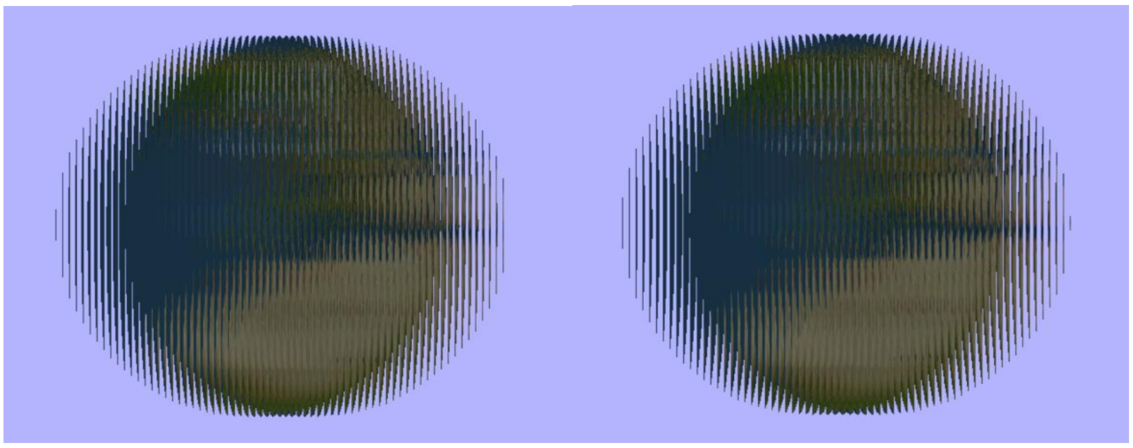


Figure 6.1: ***Training Globes*, 3-D integral images generated with a third method.**

- **Group 1** is a group of the images that are generated with the method *DIVGL* and include a short video called (***Rotating cube***), 3 static images called (***Axe man 1***), (***Cutlery***) and (***globe***), and shown in Figure 6.2, Figure 6.3, Figure 6.4, and Figure 6.5 respectively.

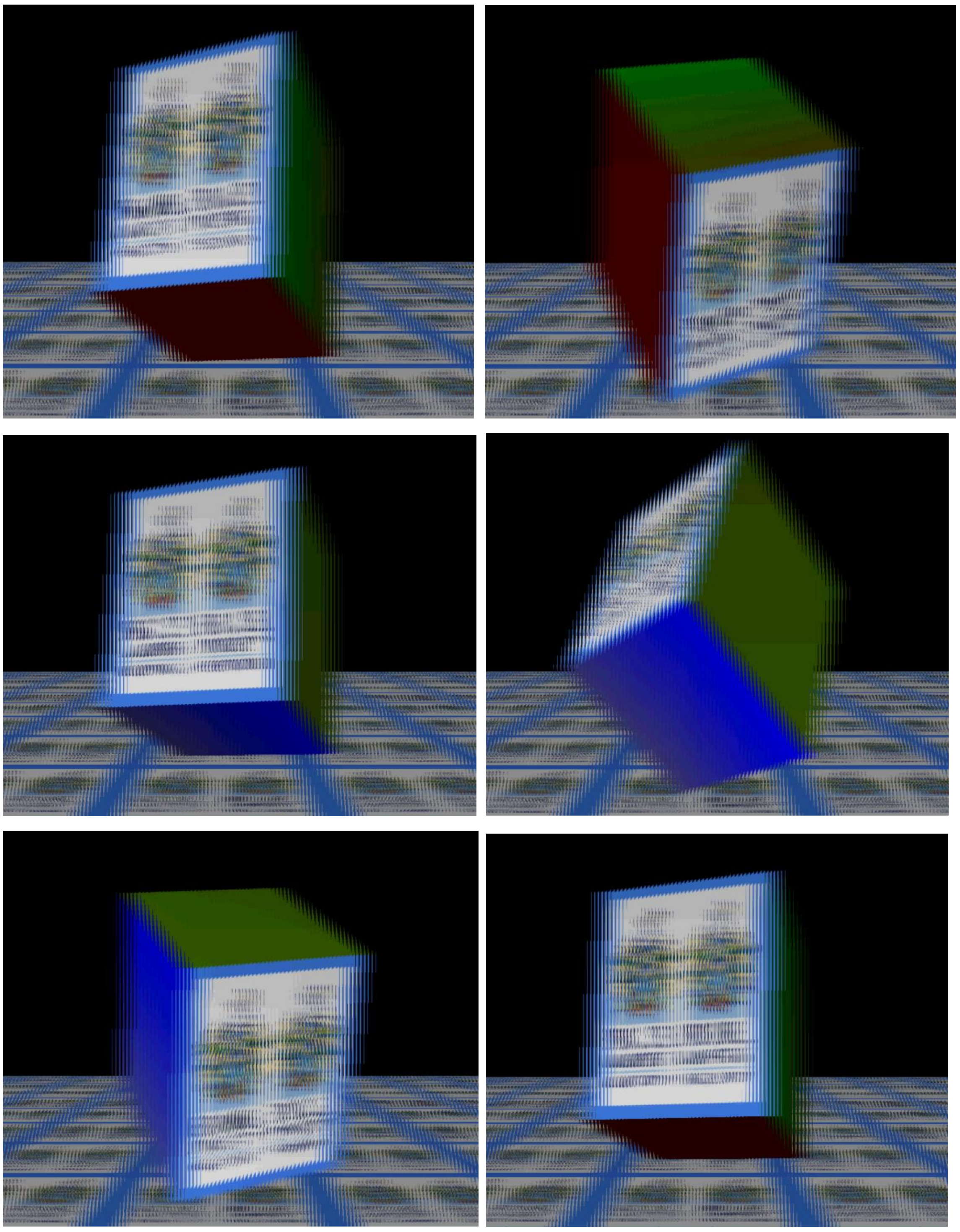


Figure 6.2: *Rotating cube*, a 3-D integral images video generated with *DIVGL* method.

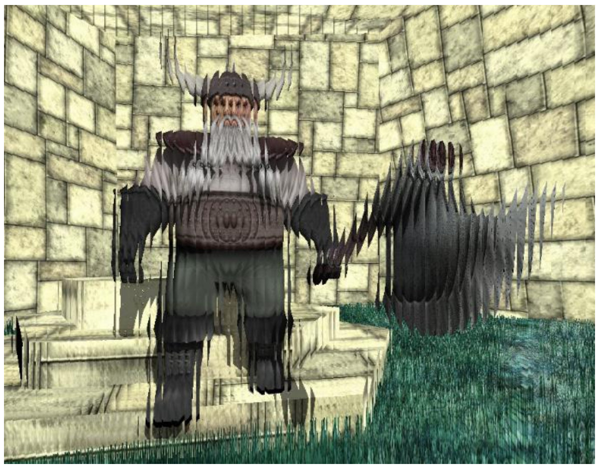


Figure 6.3: ***Axe man 1*****, a 3-D integral image generated with** ***DIVGL*** **method.**

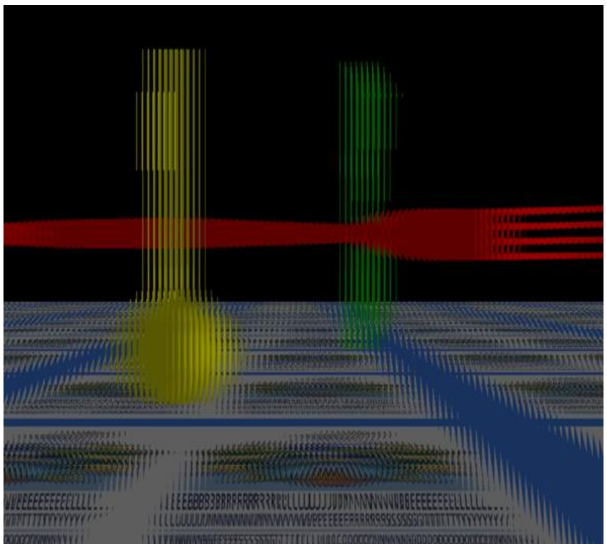


Figure 6.4: ***Cutlery*****, a 3-D integral image generated with** ***DIVGL*** **method.**

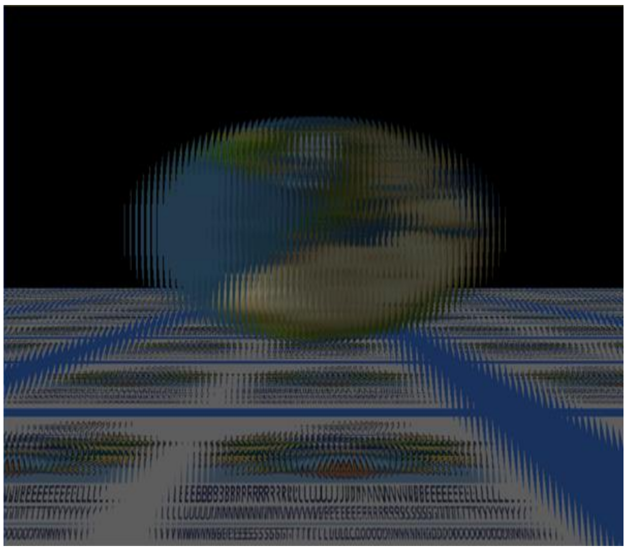


Figure 6.5: ***Globe*****, a 3-D autostereoscopic integral image generated with** ***DIVGL*** **method.**

Group 2 is a group of the images that are generated with the method *DCTarget* and include a short video called (***Angel in the castle***), 3 static images called (***Boy***), (***Angel and warriors***) and (***Axe man 2***), and shown in Figure 6.6, Figure 6.7, Figure 6.8, and Figure 6.9 respectively.

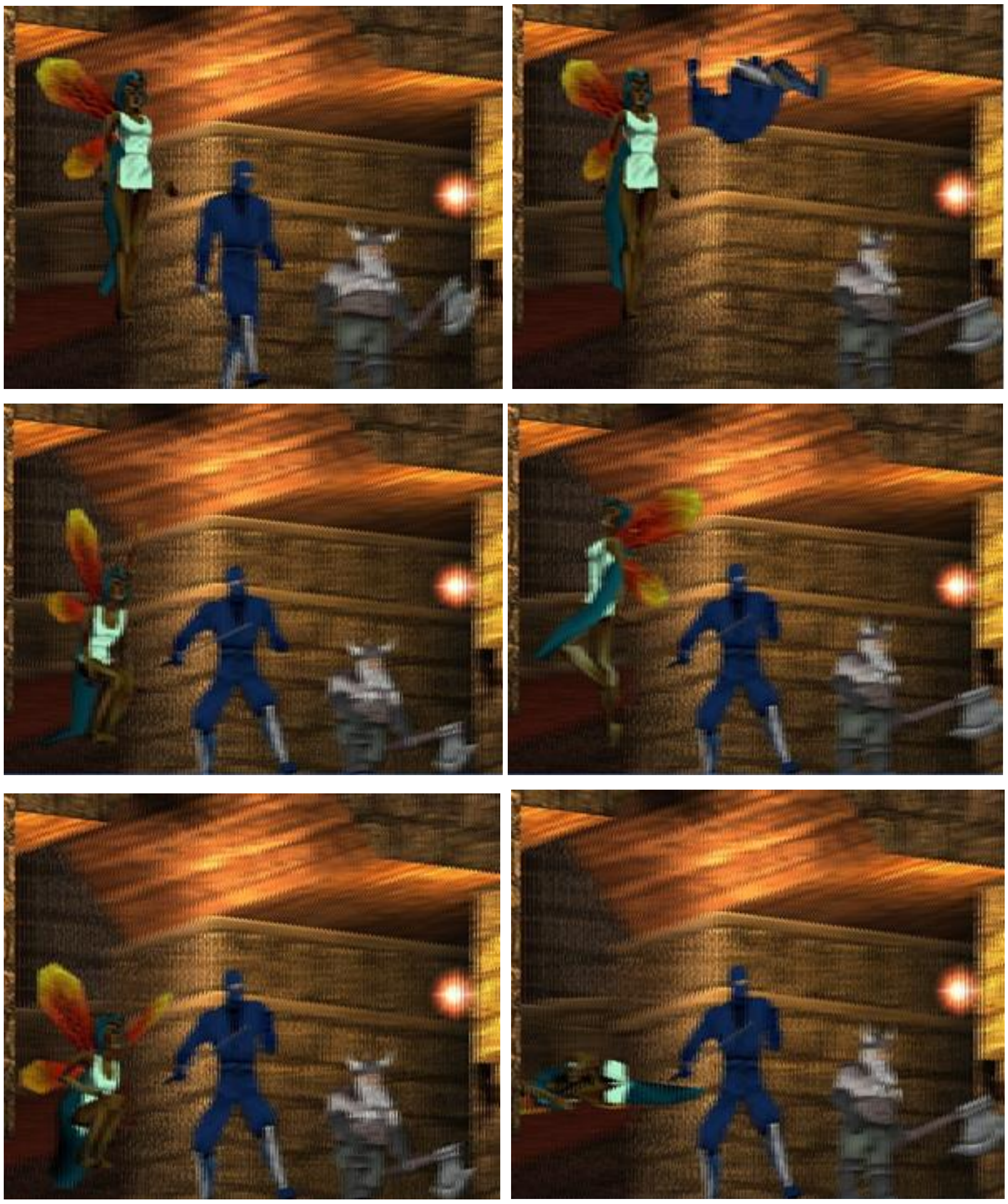


Figure 6.6: *Angel in the castle*, a 3-D autostereoscopic integral image video generated with *DCTarget* method.

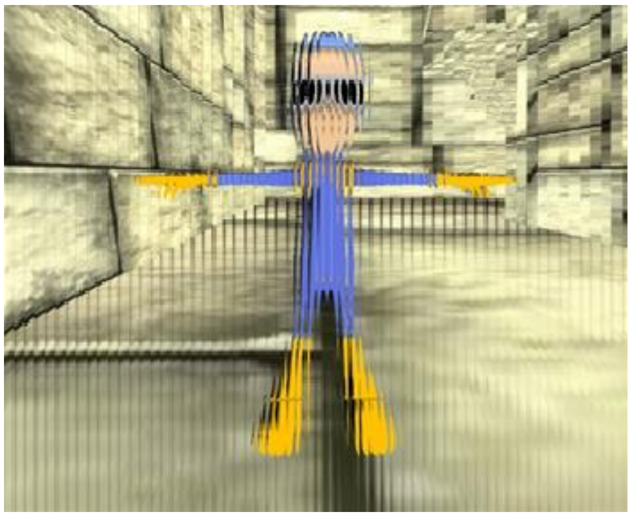


Figure 6.7: ***Boy,*** **a 3-D integral image generated with** ***DCTarget*** **method.**

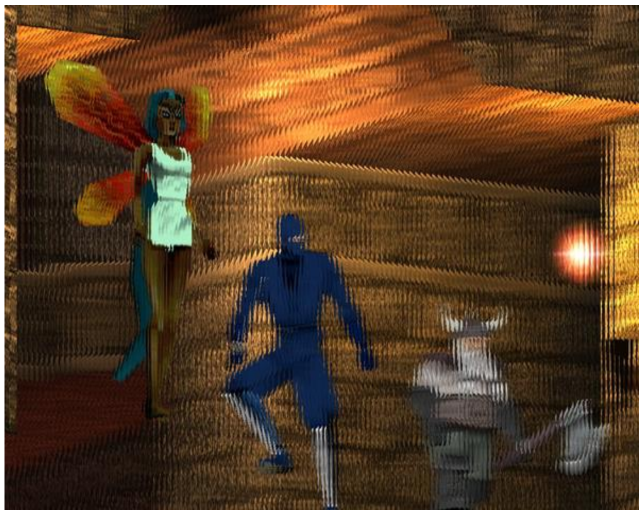


Figure 6.8: ***Angel and warriors*****, a 3-D integral image generated with** ***DCTarget*** **method.**

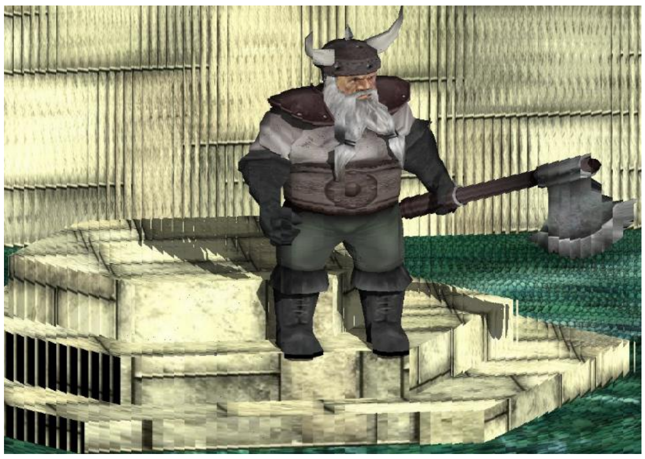


Figure 6.9: ***Axe man 2*****, a 3-D integral image generated with** ***DCTarget*** **method.**

- **Group 3** is a group of 3-D autostereoscopic integral images generated in the Interpolative Shading Technique Method (*ISTM*) [8]. Up to my knowledge, this method is the closest model, algorithm and approach to *DIVGL* method (images Group 1) and *DCTarget* method (images Group 2). Samples of an animated images (***Actor***), 5 static images called (***Ghost***), (***Pots***), (***Network***), (***Ballerina***) and (***3-Tee***) and shown in Figure 6.10, Figure 6.11, Figure 6.12, Figure 6.13, Figure 6.14, and Figure 6.15 respectively.

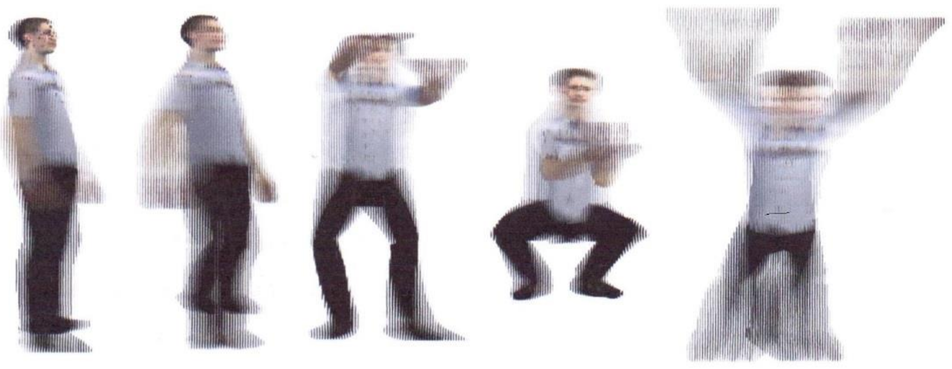


Figure 6.10: ***video Actor*, 3-D integral images generated with *ISTM* method [8/page 131].**

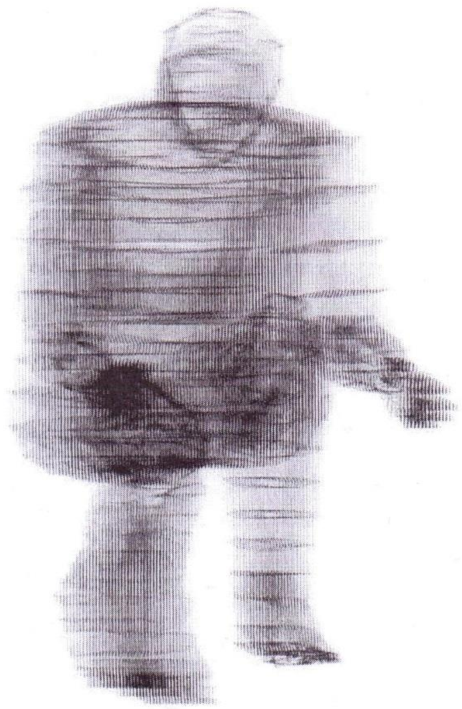


Figure 6.11: ***Ghost*, a 3-D integral image generated with *ISTM* method [8/page 47].**

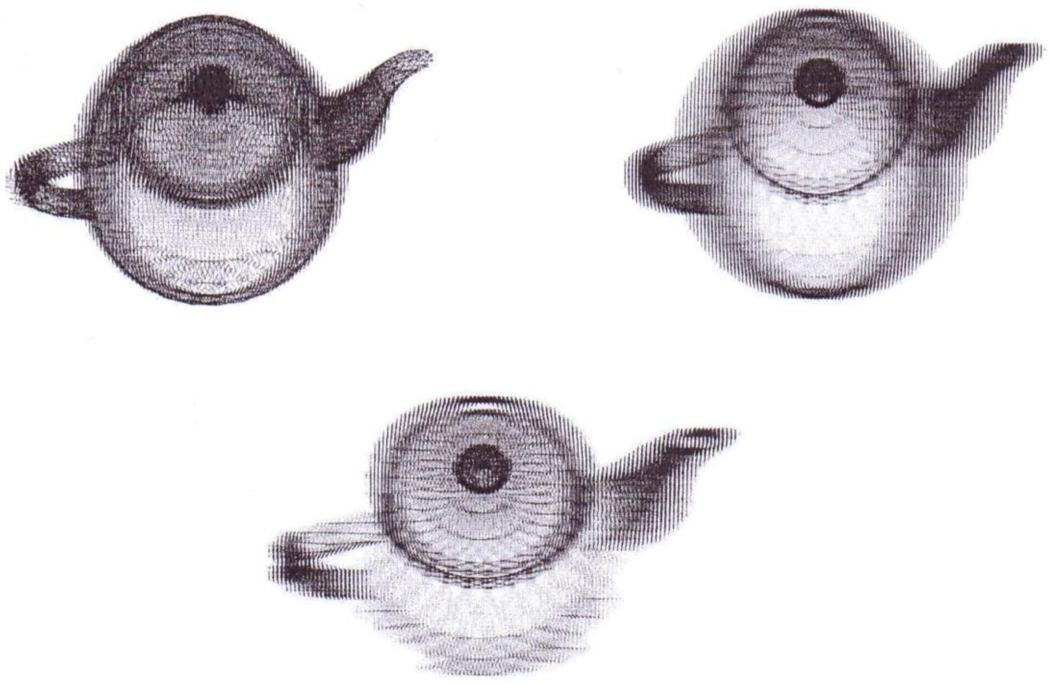


Figure 6.12: ***Pots*****, 3-D integral images generated with *ISTM* method [8/pages 58-64-67].**

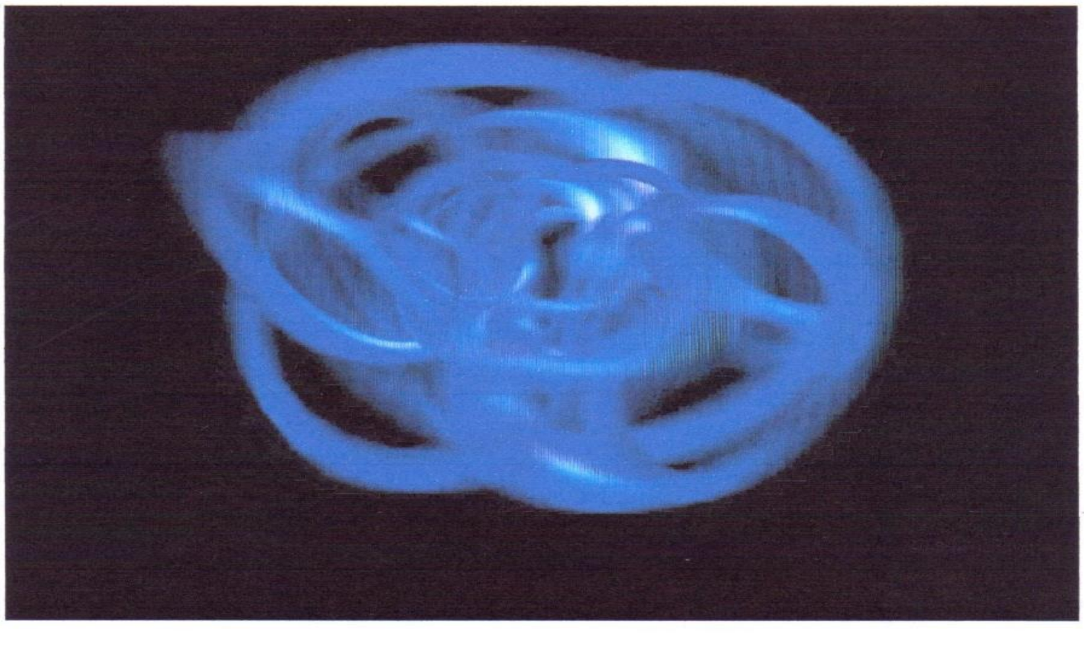


Figure 6.13: ***Network*****, a 3-D integral image generated with *ISTM* method [8/page 126].**

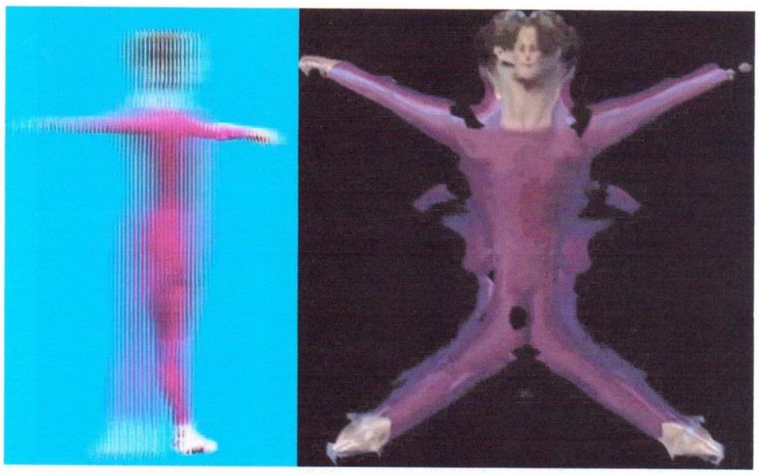


Figure 6.14: ***Ballerina*****, a 3-D integral image generated with *ISTM* method [8/page 131].**

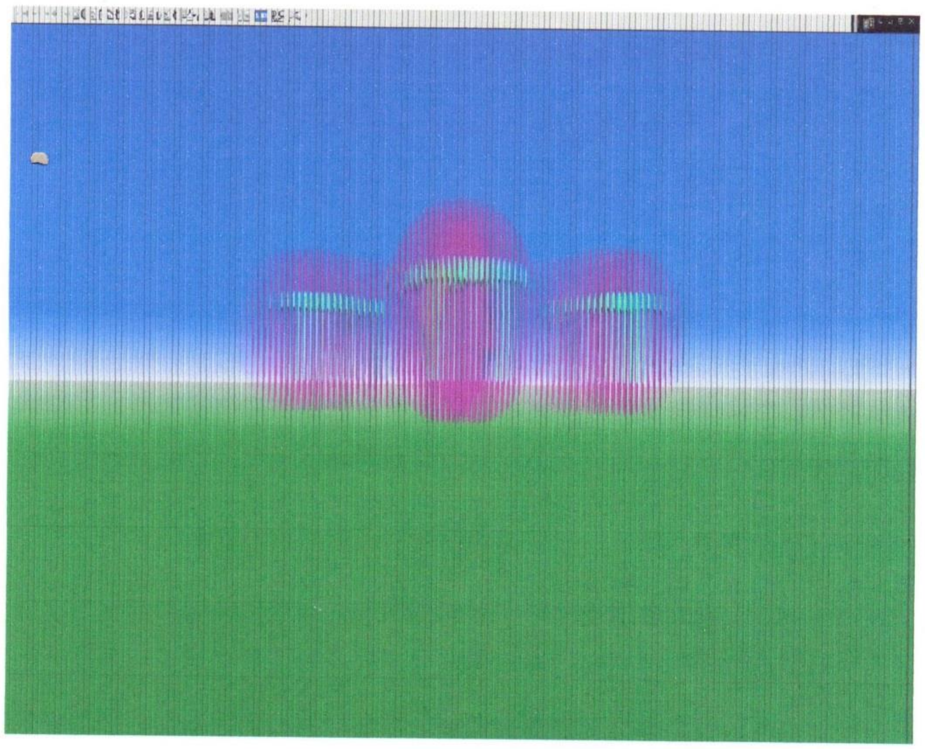


Figure 6.15: ***3-Tee*****, a 3-D integral image generated with *ISTM* method [8/page 143].**

Display device: a normal Laptop with an LCD screen with 13.5 X 7.5 inches and resolution of 1024 X 768 dpi mounted by a cylindrical lens array with a lens pitch of 2 mm, a focal length of 3 mm, and dimensions of 11.3 X 7 inches.

Viewing distance: The quality of integral images should be independent to the viewing distance. Therefore, the viewers have the freedom to move backward or forward in front of the display screen while viewing the integral images. The quality of experience should not have any degradation or improvement as a result of changing the viewing distance. However, in the case of viewing a normal TV, it is advised that the distance to the display screen should be related to the size of screen so that the viewing angle is about 30 degrees. This distance optimizes the comfort the viewer experience when viewing the images. In the test of comfort during this subjective assessment, the viewers should maintain a distance to the display screen so that the viewing angle is approximately 30 degrees. In other tests the distance is up to the viewer who should try to change the distance freely with any effect to the quality of experience.

Test location: the display and test should be taken in a comfortable room by one or more of the viewer at the same time providing no interaction or effect between each other that can distract the test or affect the results. The room should be dark enough to allow the viewer to view the image with the internal light of the display device (i.e. computer).

The questionnaire, test instructions, and user feedback: at the beginning, viewers will be given written hand-outs including a short questionnaire to collect demographic information about the viewers, Appendix 1, the test instructions, and the questionnaire on which the score of the rated images will be recorded. Appendix 2 includes two parts, the first part is the test instructions that have to be given to viewers to guide them through the test process, and the second part is the test. A questionnaire called Feedback Appendix 3 to be given to participants after the actual test to have an idea about their opinions in the 3-D technology in general, 3-D autostereoscopic images, integral images, the application software that is introduced to generate integral images, the introduced methods of generating integral images, and the assessment in which they are going to participate.

Viewers: the tests took place in an industrial research institution in London and the test was extended to include participants from other institutes. Viewers are asked to participate in the test individually or in small groups such as 2 or 3 viewers at the same time providing the number of viewers does not affect the results or the conditions of the test. The total number of participants is 23. Table 6.1 shows the demographic data the participants stated about their selves.

Categories			Number	Percentage
Gender	**Male**		15	65.2%
	Female		8	34.7%
Age	**Under 20**		0	00.0%
	From 20 to 30		6	26.0%
	From 30 to 40		10	43.5%
	From 40 to 50		5	21.7%
	50+		2	08.7%
Education	**PhD** **(or PhD student)**		13	65.5%
	Master **(or Master student)**		5	21.7%
	Degree **(or Degree student)**		3	13.0%
	College		2	08.7%
	School		0	00.0%
Occupation	**Manager**		5	21.7%
	Researcher		8	34.7%
	Employee / Engineer		9	39.1%
	Full time student		1	04.3%
	Unemployed		0	00,0%
Using experience **(How often do you watch 3-D content)**	**Once + a week**		3	13,0%
	Once a month		6	26.0%
	Several times a year		9	39.1%
	Once a year		4	17.4%
	Less often		1	04.3%
	Never		0	00.0%
Vision **(Do you wear glasses or contact lenses to see at far or at near)**	**No**		9	39.1%
	Yes	**At far**	8	34.7%
		At near	6	26.0%
Professional experience **(Experience of the viewer in 3-D imaging or image processing)**	**Good experience**		6	26.0%
	Moderate experience		12	52.2%
	Some experience		4	17.4%
	No experience		1	04.3%

Table 6.1: the demographic data the participants have shared.

6.1.3. The Test description

At the beginning of the test, a training session will take place. The training session is aimed to declare the mentioned five criteria the assessment is questioning about. The images shown in Figure 6.1 will be displayed and exposed to the viewers and the following concepts will be explained:

- **Comfort**: the viewer should focus on the static images or videos for few minutes, if the viewer feels of any fatigue or tiredness in the vision, then the image is causing one of the worst disadvantages in the 3-D images. The worse the vision fatigue is the lower rate or score in this criteria should be given to the image in question when the real test is carried out.

- **Crosstalk**: when the viewer moves to the left or right in proportion to the 3-D image, the viewing angle changes and therefore, the image will be seen from another perspective and the sub-image seen by the viewer's eye will be switched from one sub-image to another. The smooth switching from one sub-image to the other is one of the advantages of the 3-D integral images. The viewers should move right and left so that they judge the smoothness of the switching. The switching smoothness should be judged by the viewer in the actual test so that a higher score is given to the image crosstalk of the evaluated image.

- **Parallax**: the 3-D autostereoscopic integral images generated with the introduced methods can provide either horizontal parallax or full parallax. Full parallax that includes horizontal and vertical parallax can be viewed when using a spherical microlens array. Horizontal parallax can occur when cylindrical lens array is used which is the case of our test. Only horizontal parallax will be evaluated in this test. In the test session, viewers will be asked to move right and left while watching the integral videos and images and show them how some parts of the image is hidden when looking on the image from one perspective while that part can appear when moving to a different position and vice versa. In the actual test, the score of parallax criterion will be higher when the horizontal parallax is more noticeable and clearer to the viewer. For example, in the test image, parts of the globe on the extreme left or right will be hidden and these parts will appear when the viewer moves to the left or right respectively.

- **Depth**: 3-D images are distinguished from the 2-D image by several features, the depth perception in one of the most important features of 3-D images. In the best scenario, viewer will feel of the depth on both sides of the display screen. The quality of the image will be highly dependent on the depth of the image. Objects in the scene can be viewed as if they are located in the deep depth of the screen or flying away of the display screen towards the viewers. Viewer will judge the depth based on the amount of depth or the virtual distance that can be felt in the image. When the image provides the feeling of a higher depth, the score of depth for the evaluated image will be higher. In the training session, the left globe is meant to be fully displayed behind the display screen, while the right globe is rendered so that it looks fully in front of the display screen or the laptop screen in our case. Viewer will be used on the depth concept when displaying images on both the two sides of the screen is demonstrated and the depth is defined as the virtual distance the viewer can see from the closest point to the viewer's eye to the further point from the viewer's eye.

- **Field of view**: Field of view (*FOV*) or view angle of the integral imaging system is the highest angle the viewer can see the scene when moving to the right, left, up and down in proportion to the 3-D integral image. In the training session, the viewer will be asked to move around to the left and right so that they discover the highest angle through which the scene can be seen. The score of quality in terms of *FOV* is proportionally dependent on to the angle of view because the higher *FOV* provides more freedom in moving and wider space the viewer can view the image and more viewers can see the same display at the same time. In the actual test, the viewer will be asked to discover this feature in the evaluated images and videos and score this criterion for each image and record the score on the results table.

Viewers will evaluate the five criteria for each image of the three groups of images. The score should reflect the general opinion of the viewer about each criterion for each image in the group. The score is expressed as a percentage, then the viewer should rate the criteria with an integer number between 0 and 10, so that 0 against a specific criteria for a specific image means absolutely unsatisfied and the quality of the image in terms of that criteria is null, Whereas, 10 means absolutely satisfied with the quality of that image in terms of that criterion. The viewer are given the following instruction list in which the method of scoring the image is explained:

In this test, you will have to judge the quality of both the static and the animated 3-D autostereoscopic integral images.

- The images and videos will be displayed on this laptop mounted by a cylindrical lens array in front of you and the voting will be performed on the following pages.
- Three groups will be displayed and rated: groups 1 and group 2 each one includes 3 static images and one video, and group 3 includes 6 images.
- Group 1 will be displayed first followed by group 2 and then group 3.

- The video in each group will be displayed continuously for maximum of 5 minutes, whereas, each static image will be displayed once for 2 minutes maximum.
- Please stay focused during the test and feel free to move slightly left and right, backward and forward to judge the five criteria we explained in the training session.
- Two minutes are given after each display to fill in the form stating your opinion scores you worked out during the display. Write them next to the criterion you are scoring in the field specified for that image/video. However, the scores can be added during the display time.
- 5 minutes break will be given between the groups to finalize groups 1 and 2 scores.
- 5 minutes will be given to read these test instructions.
- 10 minutes at the end will be given to finalize the scores, fill in the demographic data form, and fill in the feedback form.
- The sequence of the test activities will be as follows: 5 minutes to read these instructions, 5 minutes video 1/group 1 display, 2 minutes break, 2 minutes static image 1/group 1 display, 2 minutes break, 2 minutes static image 2/group 1 display, 2 minutes break, 2 minutes static image 3/group 1 display, 5 minutes break, 5 minutes video 2/group 2 display, 2 minutes break, 2 minutes static image 1/group 2 display, 2 minutes break, 2 minutes static image 2/group 2 display, 2 minutes break, 2 minutes static image 3/group 2 display, the same is repeated for group 3, 10 minutes break making the total of 83 minutes. However, the time can be much shorter if the decisions were taken faster.
- The three groups have 3 result tables.
- Each image/video of the three groups has a special column in the result tables. Group 1 and group 2 have the same result table, whereas, group 3 has a separated result table.
- Each criteria for the image/video has a special cell in the related image/video field.
- The criteria are the five criteria mentioned above: Comfort, Crosstalk, Parallax, Depth, and View Angle.
- Each cell will contain the Opinion Score of the quality of the corresponding image in terms of the related criteria.

The score can be any integer number between 0 and 10, where 0 indicates (bad), and 10 indicates (excellent).

Figure 6.16 shows an example of the images displayed in the assessment and the devices used. The words (*Brunel University West London*) can be read clearly in the image after using the display device.

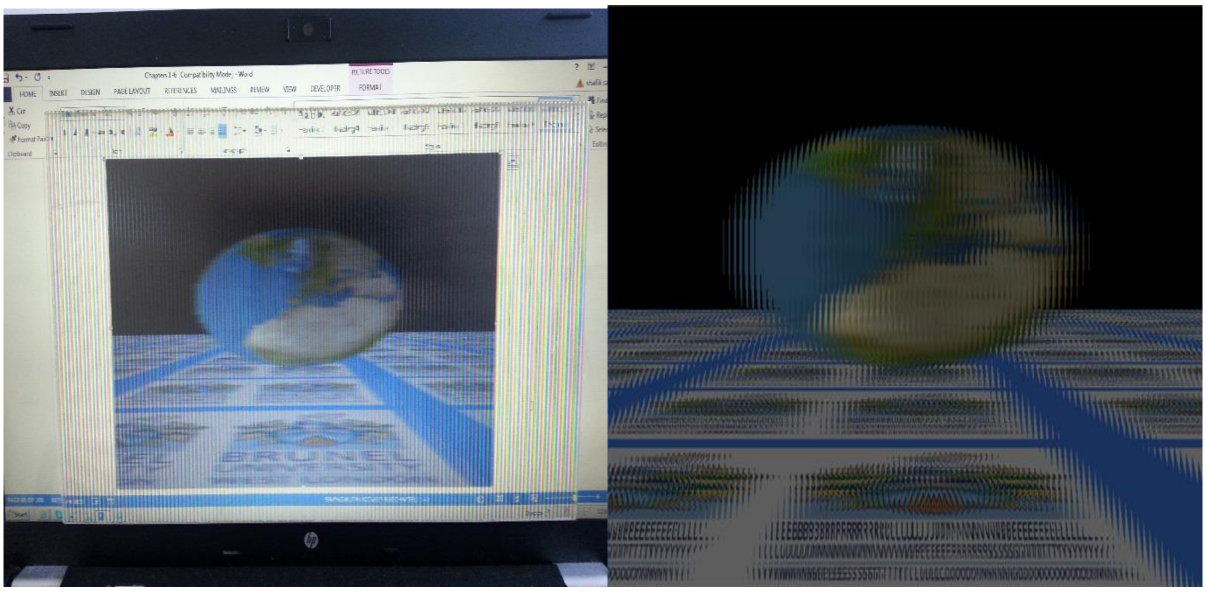


Figure 6.16: An example of the images displayed in the assessment.

6.1.4. Analysis and results

The total number of participants is 23, the viewer's opinions of the 5 criteria of each image and the calculations derived from the results are stated in Appendix 4. Figure 6.17 shows the Root Mean Square (*RMS*) of the average opinions of the criteria given by each viewer for each one of the three methods. Each viewer stated his/her opinion about the quality of experience in terms of each criteria of the five mentioned criteria of each image of the images grouped in three groups. In order to calculate the numerical evaluation of each method based on the opinion of the viewers, the average of each criteria was taken for all the images belonging to the same group. For example, the comfort criteria of group 1 images were averaged (i.e. added and divided by four the number of images).

The averaging result can be considered as the evaluation of the comfort criteria for the four images of group 1, in other words, the comfort evaluation of *DIVGL* method with which group 1 images were generated. The same averaging process was applied to all the criteria of each group. *QoE* for each group was considered as the Root Mean Square RMS of the five calculated criteria referred to in the graph of Figure 6.17 as OPINIONS RMS.

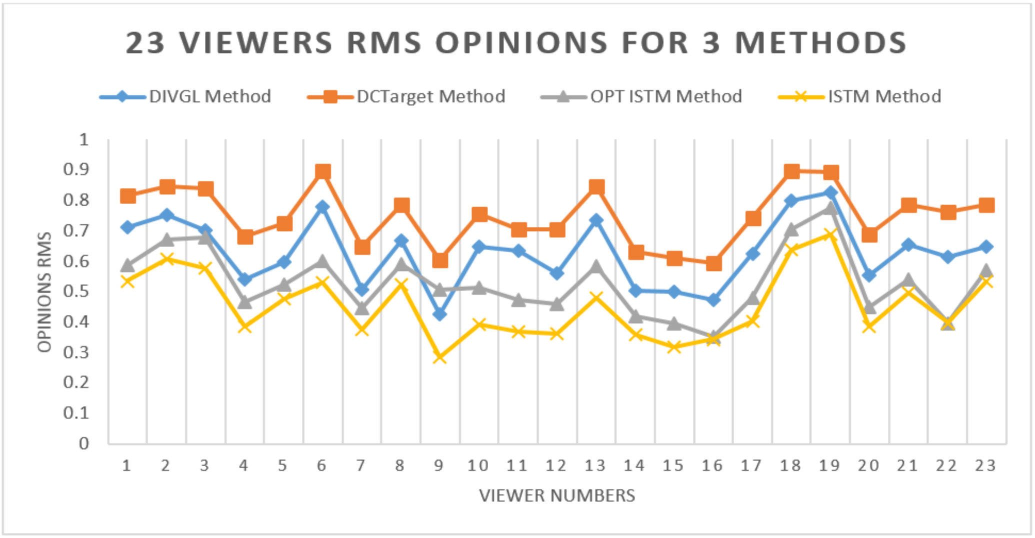


Figure 6.17: the opinions average *RMS* of 23 viewers for the three methods.

Opinion RMS was calculated for each viewer about each method of the studied methods based on Equation 6.4. The number of images chosen for methods *DIVGL* and *DCTarget* are 4, while, *ISTM* method images number is 6. The results of *ISTM* method were optimised by selecting the best results of four images and considered to calculate the *QoE* of the method whereas, the results of the other two images were discarded. In Figure 6.17, the optimized *RMS* opinions of *ISTM* method is referred to as *OPT ISTM* method.

Each score of the scores given by the participants are normalized by 10, therefore, the *RMS* were portions of 1. The *QoE* calculated based on the viewer's opinions for *DIVGL* method was higher than that of *ISTM* method and the optimized *ISTM* method and therefore, the quality of *DIVGL* method is higher than the quality of *ISTM* method, whereas, the quality of *DCTarget* method is even higher than both of the two methods *DIVGL* method and *ISTM* method.

If the opinions *RMS* that is calculated for each method is considered as the viewer opinion of that method, then *MOS* of the method can be calculated by averaging the method opinion *RMS* of all the viewers, in other words, adding the *RMS* for each method that was calculated for each viewer and divide the result by 23 which is the total number of viewers. *MOS* of the three methods calculated by averaging the opinion *RMS* of the viewers is shown in Table 6.2. To compare *QoE* for the 3 methods, *QoE* is considered as a percentage to the highest *MOS*:

	DIVGL Method	**DCTarget Method**	**Opt ISTM Method**	**ISTM Method**
MOS	0.628549	0.750213	0.530293	0.454755
QoE	84%	100%	71%	61%

Table 6.2: a comparison between the three methods and the optimized ISTM.

If the opinions *RMS* for each image was considered as the viewer opinion of that image, then *MOS* of the image can be calculated by averaging the opinion *RMS* of all the viewers, in other words, adding the *RMS* for each image calculated for each viewer and divide the result by the total number of viewers (i.e. 23). Figure 6.18 shows a comparison between the *MOS* results of the images used in the assessment. MOS is calculated by averaging the *RMS* of the images. The comparison shows DCTarget *QoE* as the highest *QoE* amongst the three groups of images. *DIVGL QoE* ranked second exceeding *ISTM*.

Figure 6.19 represents the *MOS* of each criteria quality for each image of the images used in the assessment based on the viewer's opinions. The graph shows that the images belonging to *DCTarget* method have the highest *MOS* for each image and for almost every criteria in comparison with the other two methods, while, *DIVGL* method has higher *MOS* than *ISTM*.

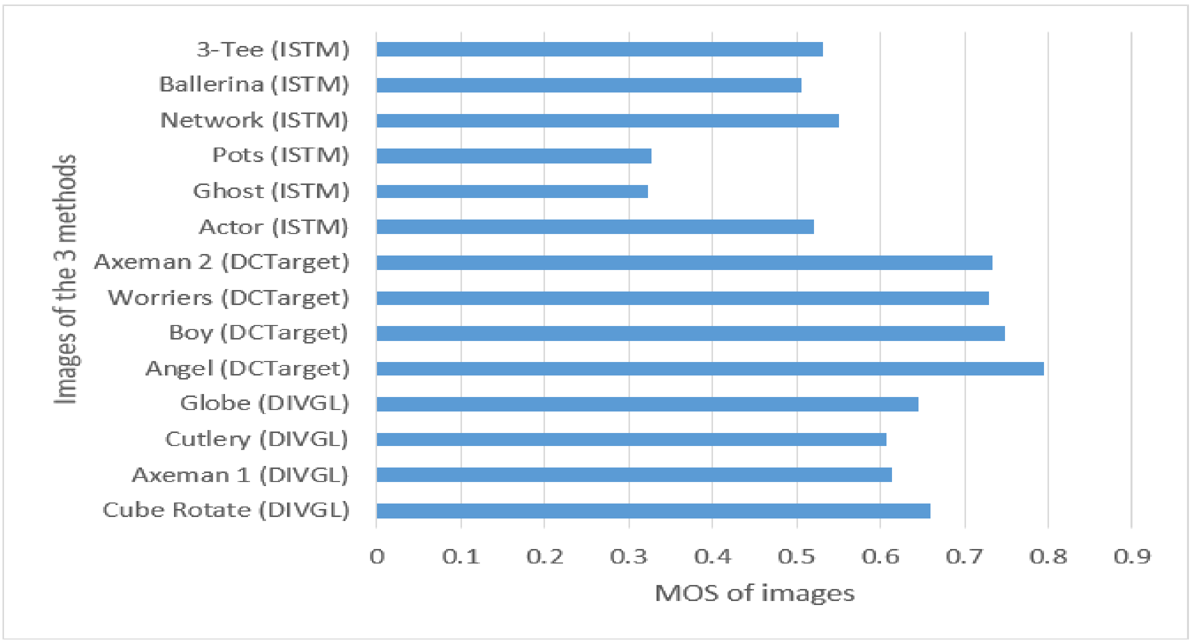


Figure 6.18: *MOS* of each image of the 3 groups.

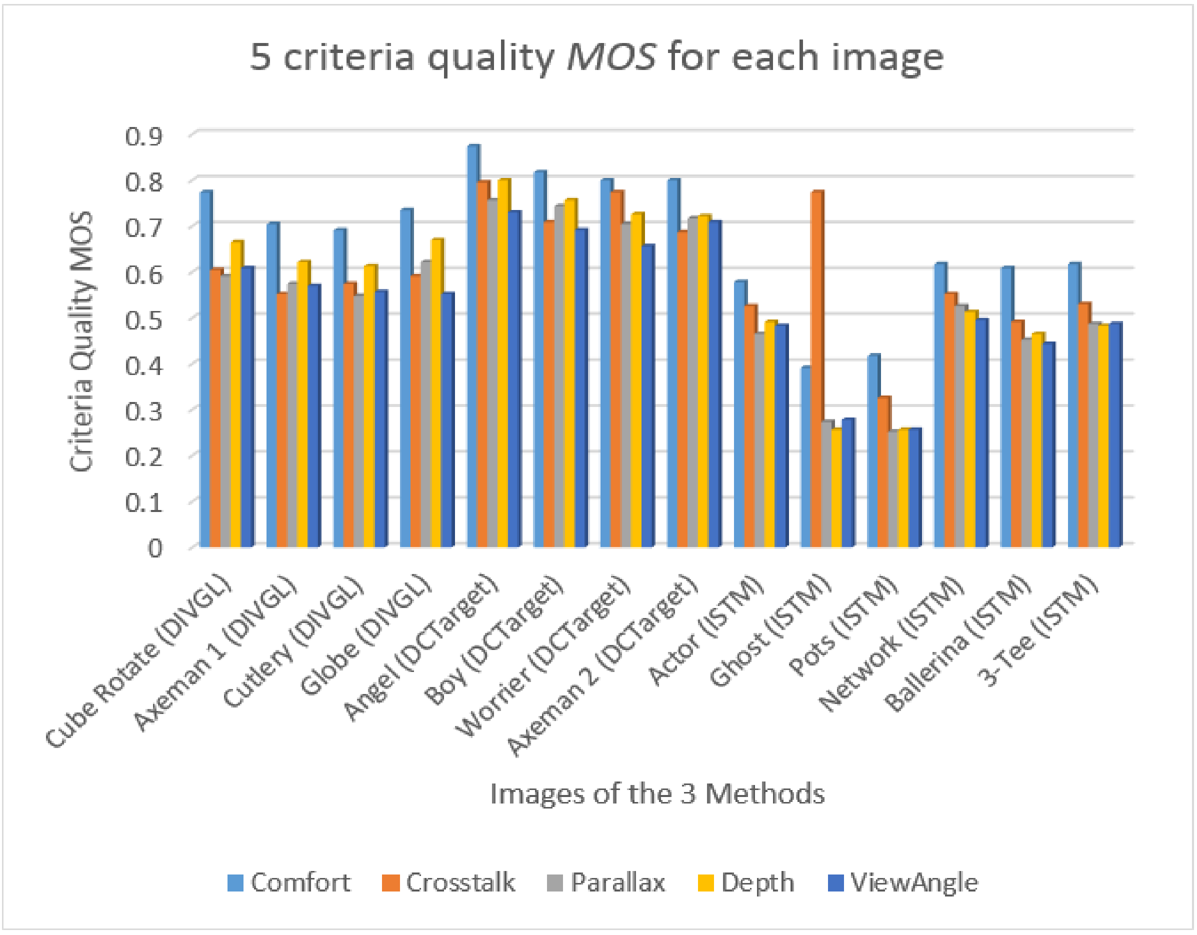


Figure 6.19: *MOS* of each criteria for each image of the 3 groups.

On the other hand, if *MOS* of every criteria for each image was calculated (i.e. the average of all the opinion scores for each criteria apart), the *QoE* can be considered as the *RMS* of all the resulting *MOS* for each image.

Figure 6.17 shows the calculated *RMS* of the average scores of the images for each criteria. In theory, the average of the *RMS* for the images belonging to the same method should be equal to the *RMS* of the average scores for the criteria of these images. In other words, previously, the scores of each criteria were averaged, then the *RMS* of the averaged scores was calculated for the images of each method, Figure 6.17 represents the results for each viewer. Now, the *RMS* of the criteria of each image is calculated and then the resulting *RMS* are averaged for each method to be compared with the results shown in Figure 6.17. Figure 6.20 shows the calculated average of the *RMS* values for the images of each group in addition to the optimised group that contains the best scored 4 images of *ISTM* method images. In comparison with Figure 6.17, the graphs are almost identical to the graphs plotted and shown in Figure 6.17. The two sets of data are plotted and represented in the chart so that the difference between the two sets of data is shown to represent the error of calculation that can be considered as the error in considering the assumption that the *QoE* is proportional to the *RMS* of the particular opinions of the criteria.

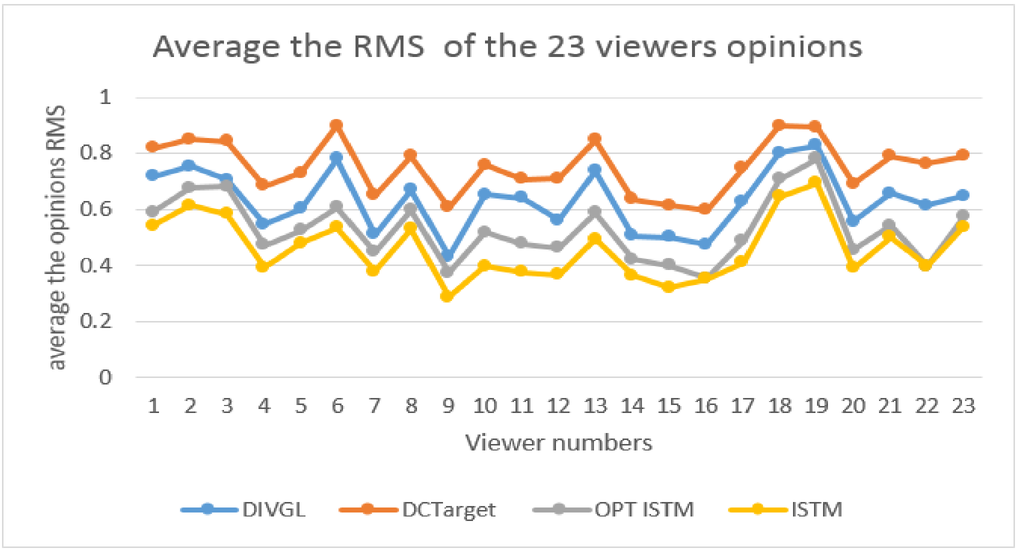


Figure 6.20: the average *RMS* of the opinions of 23 viewers about the three methods.

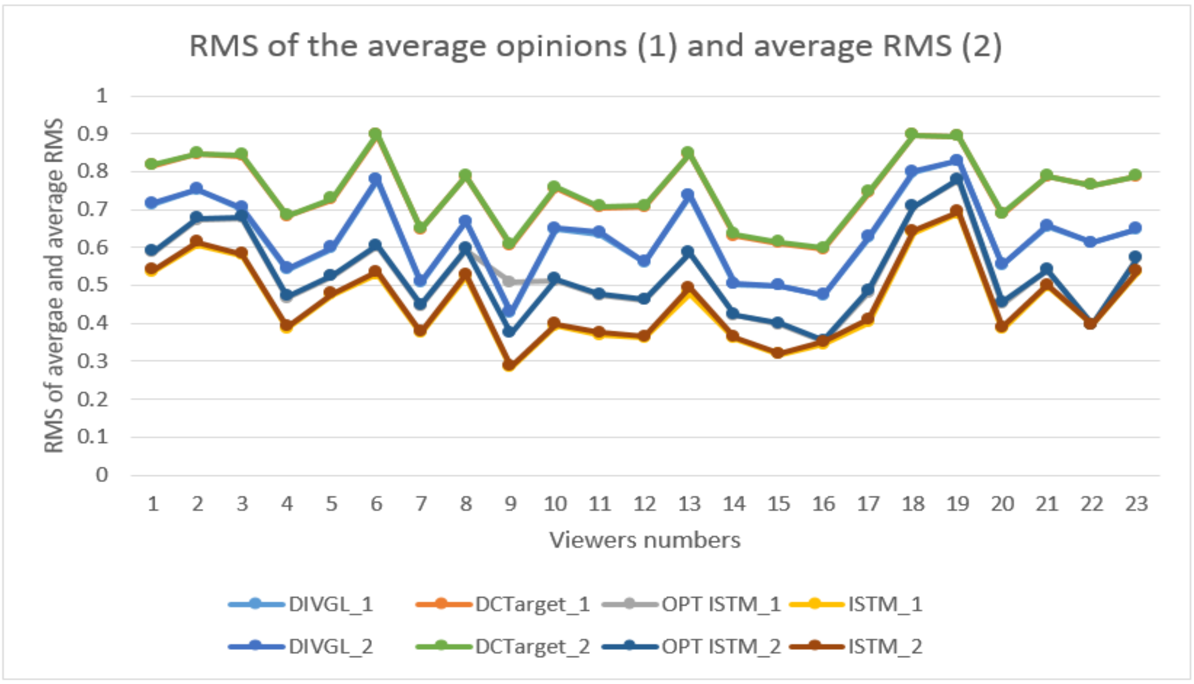


Figure 6.21: the average *RMS* of the opinions and the *RMS* of the opinions combined.

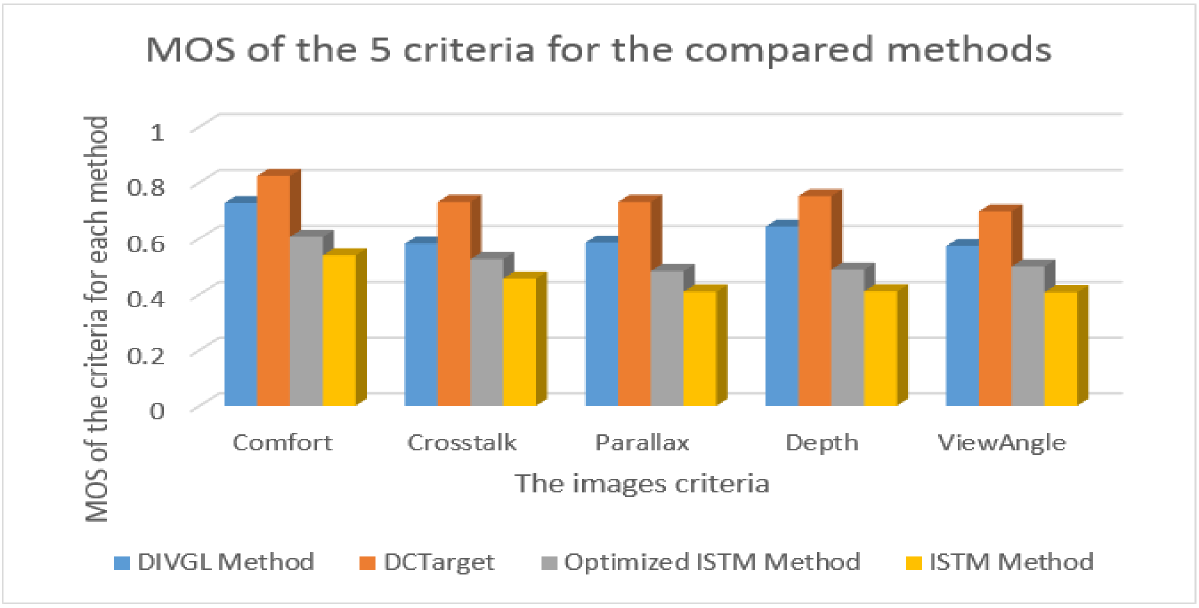


Figure 6.22: ***MOS*** **of the criteria scored by viewers for each method.**

Figure 6.21 shows the combined two ways of calculations, the first is the one when the average of the criteria opinions are calculated and the *RMS* of the average was calculated and the second is the one when the *RMS* of the criteria of each image from each group were averaged. The two ways are aimed to calculate a numerical value to reflect the *QoE* of each method of the examined 3 methods considering the best 4 images from group 3. The graphs show that the results of the two ways are almost identical with a very small error caused mainly by the approximations conducted in the calculation process.

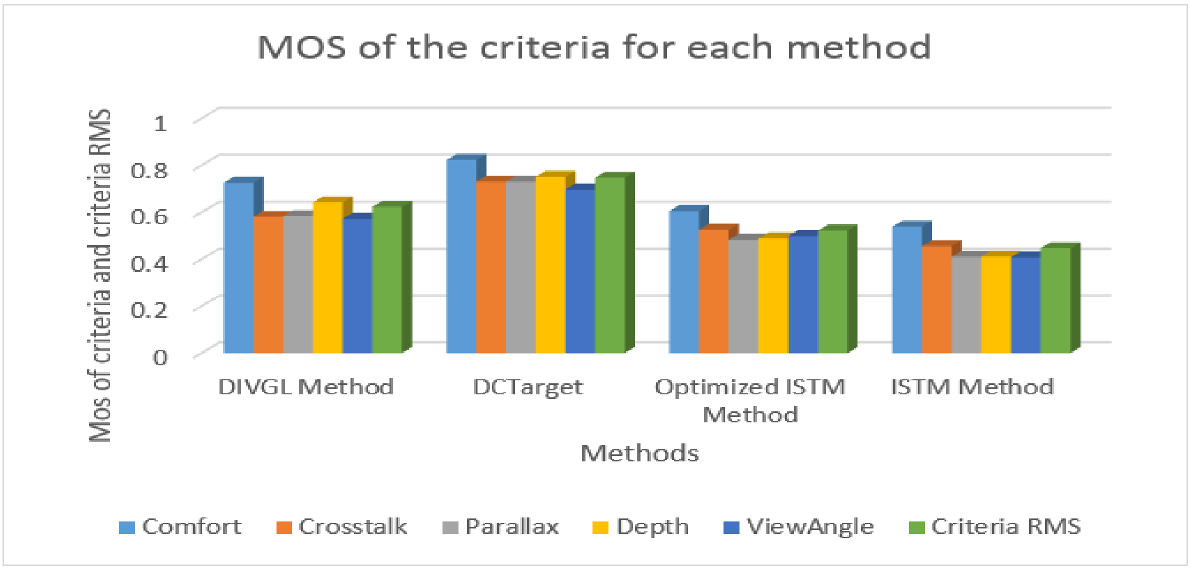


Figure 6.23: ***MOS*** **and** ***RMS*** **of the criteria for each method.**

Figure 6.22 shows a comparison between the compared methods about the criteria by comparing the *MOS* taken as the average of the opinions provided by the viewers about each criteria. It is clear that *DCTarget* method scored the highest *MOS* for all the criteria.

Figure 6.23 represents the methods with the *MOS* of the scored criteria and the *RMS* of these criteria for each method showing *DCTarget* method with the highest scores in comparison with *DIVGL* and *DIVGL*'s are higher than those of *ISTM*, whereas, the optimized *ISTM* with the best images considered scored lower than *DIVGL* and higher than *ISTM* method with all the tested images taken into consideration.

6.1.5. Viewers Feedback

The numerical average (*Na*), and the weight percentage (*Wp*) are calculated with the equations:

$$Na = (\sum_{i=1}^{M} VNi * Wi)/N \qquad (6.6)$$

$$Wp = Na/MaxW \qquad (6.7)$$

where, *i* is the index of the selected choice, *M* is the maximum number of choices, *VNi* is the number of viewers who selected the choice *i*, *Wi* is the weight of the choice, *N* the total number of viewers, *MaxW* is the maximum weight of the choices. The maximum weight for all the questions was 4 and the total number of viewers is 23, the number of choices was 5 for all the answers in this feedback from. For example, the first question average is [(12×4) + (8×3) + (3×2) + (1×0) + (0×0)]/23 = 3.39, and the percentage = 3.39/4 = 84.7% which means that the importance of 3-D technology is rated by the viewers as 84.7%. Table 6.3 presents the feedback provided by the viewers and the associated calculations.

Question		**Viewers Number**	**Numerical Average**	**Percentage**
What is your opinion in 3-D technology and 3-	**Very important (4)**	12	3.39	84.7%
	Important (3)	8		
	Reasonable (2)	3		

D imaging	**Less important (1)**	0		
	Unimportant (0)	0		
What is your opinion in 3-D autostereoscopic imaging	**Very important (4)**	9	3.22	80.4%
	Important (3)	10		
	Reasonable (2)	4		
	Less important (1)	0		
	Unimportant (0)	0		
What is your opinion in 3-D integral imaging and its characteristics	**Very important (4)**	5	2.91	72.8%
	Important (3)	11		
	Reasonable (2)	7		
	Less important (1)	0		
	Unimportant (0)	0		
What is your opinion in the software used to produce group 1 and 2 of images	**Very good (4)**	19	3.83	95.6%
	good (3)	4		
	acceptable (2)	0		
	Less acceptable (1)	0		
	Unacceptable (0)	0		
What is your opinion in the methods of generating group 1 and 2 of images	**Very good (4)**	11	3.35	83.7%
	good (3)	9		
	acceptable (2)	3		
	Less acceptable (1)	0		
	Unacceptable (0)	0		
What is your opinion in our assessment in which you have participated	**Very simple (4)**	0	0.78	19.5%
	Simple (3)	0		
	Average (2)	5		
	Difficult (1)	8		
	Very difficult (0)	10		

Table 6.3: Feedback provided by the assessment participants.

From Table 6.3, the feedback provided by the participant in the subjective assessment show that the test procedure was difficult as the percentage of 19.5% means the test was viewed by the participants to be tending to difficulty. The main reason for that can be referred to the difficult concepts of the quality criteria that the *QoE* was examined based on them.

6.2 COMPLEXITY AND SPEED OF RENDERING EVALUATION

Complexity and speed of rendering are key factors in the computer graphic functions. In the real-time applications such as video games, complexity and speed of rendering are critical factors and the application performance is highly dependent on the speed of processing. 3-D autostereoscopic integral images are supposed to be used in the real-time applications such as video games in which the user reacts instantly with the machine; therefore, it is necessary to render and display the animated images in the speed that is needed to meet the requirements of the application. In addition to the hardware requirements such as the video card with an adequate speed and the machine on which the application is executed with an acceptable speed capacity, the software plays an important role in enhancing the application performance and improving the speed of the execution. Converting the 2-D animated images to 3-D autostereoscopic integral images should be as fast as possible to meet the speed requirements. To optimise the time that is spent on rendering the animated images, the complexity and the time that is spent on rendering a single 3-D autostereoscopic integral image should be optimized. Therefore, the complexity and the rendering speed in the different algorithms used for producing 3-D autostereoscopic integral images can be evaluated based on the complexity needed and the speed of rendering a single static image because the animated scenes are composed from a number of frames and each frame can be looked at as a single static image. An attempt to evaluate the *DIVGL* and *DCTarget* algorithms in terms of complexity and rendering speed would be introduced and explained. In addition, a comparison between the complexity and rendering speed of these algorithms and those of some other algorithms will be carried out. Up to my knowledge, an objective or subjective assessment of the complexity and rendering speed of the 3-D autostereoscopic integral images is not available and thus, more developed objective assessments are required to evaluate these images, or even, a new and different objective or subjective assessment method is required to assess the complexity and the speed rendering of the 3-D autostereoscopic integral images. The idea behind this objective assessment method is the approximate estimation of the number of arithmetic processes that are needed to render a 3-D integral image in each method of the compared algorithms. The number of arithmetic processes is supposed to be proportional to the time needed to render and display a single 3-D autostereoscopic integral image when the method is question is used and implemented with the same machine (i.e. computer). As a result, the methods are evaluated and compare against each other. The time of rendering in each method should be taken into account when selecting the suitable algorithm to use for designing and implementing a computer graphic application. In this section, a comparison on the basis of complexity and speed of rendering between *DCTarget* method and other methods including *DIVGL* method is carried out. The complexity and speed of rendering are supposed to be proportional to the number of operations and arithmetic processes that are carried out to calculate the 2D images, therefore it is proportional to the computational expenses. Up to my knowledge, the closest models, algorithms and approaches to *DCTarget* method and *DIVGL* method are the Forward Projection Pinhole Mesh Model (*FPPM*) [8], Forward Projection Finite-Sized Aperture Mesh Model (*FSAM*) [8], and Computer-Generated Integral Photography *CGIP* [48]. The procedures implemented in these methods are objectively assessed and evaluated. Figure 6.24 represents the flowchart of the integral pinhole mesh model *FPPM*. Figure 6.25 represents the flowchart of the Finite-sized aperture model *FSAM*. Figure 6.26 shows the flowchart of *CGIP* method. Figure 3.25 shows the computer generation of integral images flowchart based on *DIVGL* algorithm. Figure 4.7 shows the generation process flowchart of integral images based on *DCTarget* algorithm.

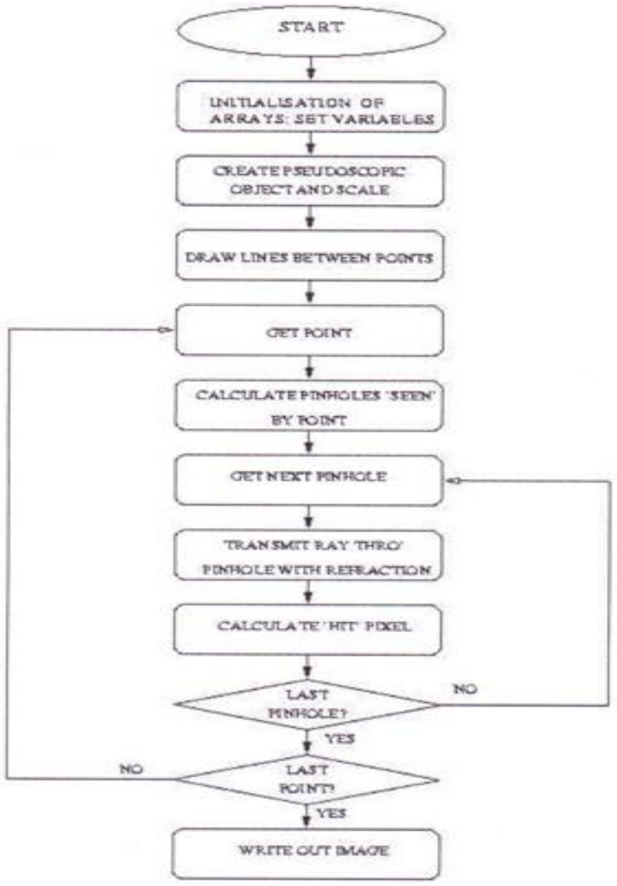

START
INITIALISATION OF ARRAYS: SET VARIABLES
CREATE PSEUDOSCOPIC OBJECT AND SCALE
DRAW LINES BETWEEN POINTS
GET POINT
CALCULATE PINHOLES 'SEEN' BY POINT
GET NEXT PINHOLE
TRANSMIT RAY THRO' PINHOLE WITH REFRACTION
CALCULATE 'HIT' PIXEL
LAST PINHOLE?
NO
YES
LAST POINT?
NO
YES
WRITE OUT IMAGE

Figure 6.24: The flowchart of the integral pinhole mesh model [8].

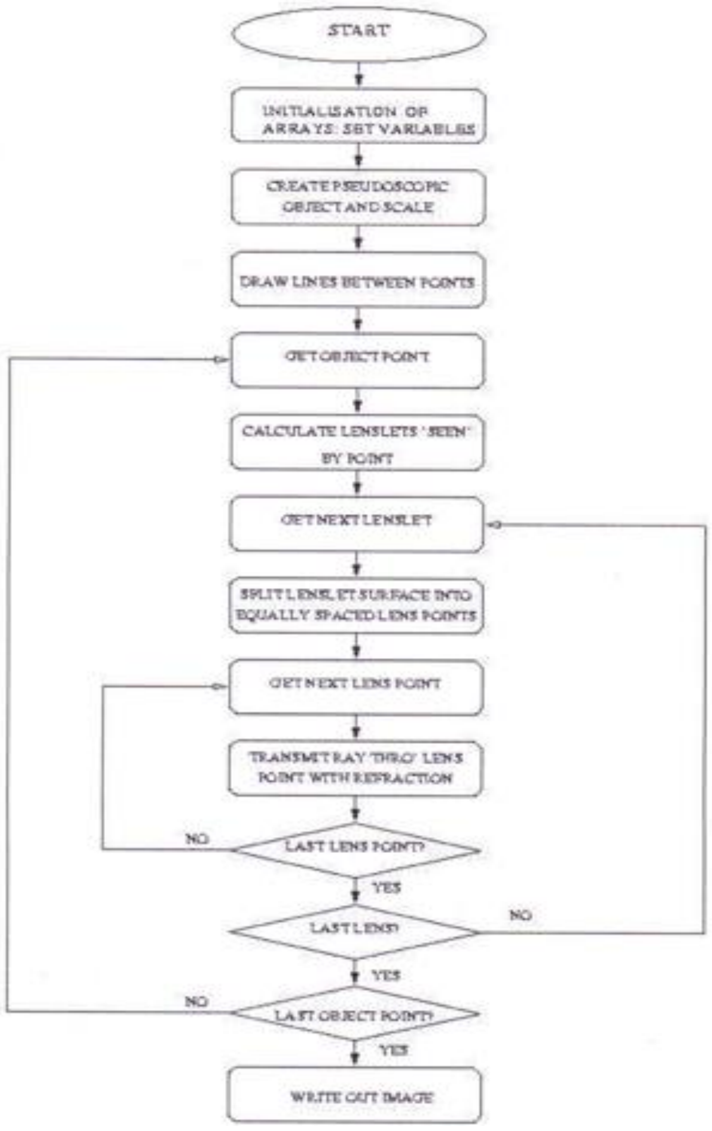

START
INITIALISATION OF ARRAYS: SET VARIABLES
CREATE PSEUDOSCOPIC OBJECT AND SCALE
DRAW LINES BETWEEN POINTS
GET OBJECT POINT
CALCULATE LENSLETS 'SEEN' BY POINT
GET NEXT LENSLET
SPLIT LENSLET SURFACE INTO EQUALLY SPACED LENS POINTS
GET NEXT LENS POINT
TRANSMIT RAY THRO' LENS POINT WITH REFRACTION
LAST LENS POINT?
NO
YES
LAST LENS?
NO
YES
LAST OBJECT POINT?
NO
YES
WRITE OUT IMAGE

Figure 6.25: The flowchart of the Finite-sized aperture model [8].

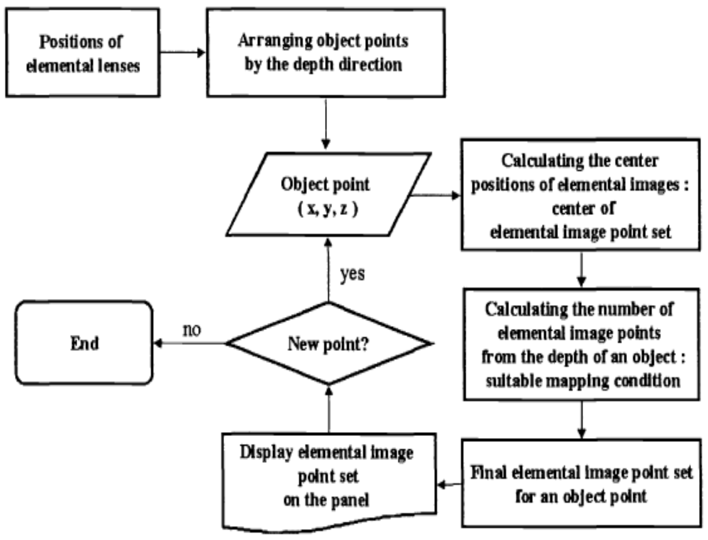

Positions of elemental lenses
Arranging object points by the depth direction
Object point (x, y, z)
Calculating the center positions of elemental images : center of elemental image point set
Calculating the number of elemental image points from the depth of an object : suitable mapping condition
Final elemental image point set for an object point
Display elemental image point set on the panel
New point?
yes
no
End

Figure 6.26: The flowchart of *CGIP* method [48].

- One of the major steps in *FPPM* and *FSAM* methods is aimed to create pseudoscopic object as it is shown in Figure 6.24. The resulting image from this method is supposed to be a simulation of the imaging process using a number of pinholes. The image formed in this case is pseudoscopic, therefore, the imaged objects should be pseudoscopic at first place so that the final image is orthogonal, whereas, this stage is not required in both *DIVGL* and *DCTarget* because the process of converting the pseudoscopic image that is resulting from the projection process to an orthogonal image is included in the algorithm. In both *DIVGL* and *DCTarget* the whole 2-D scene is imported as it is designed and generated with the computer applications such as 3D Max and the imported and converted to a 3-D orthogonal scene using these algorithms. The pixel values and locations are calculated and mapped so that the image is corrected. In order to overcome the pseudoscopic problem in *CGIP*, the object points are assumed to be a combination of planes with different depths which are arranged by the depth direction. This step adds more complexity to the methods *FPPM* and *FSAM* methods and therefore reduces the speed of rendering with these methods in comparison to *DIVGL* and *DCTarget* methods.
- A complicated algorithm is needed in *FPPM* and *FSAM* methods to draw lines between the object points in the scene so that the correct spatial information are captured. The triangle corners that are projected through the pinhole do not provide enough information to create the integral image. Points on the lines are then selected to be projected, the distances between these points are not equal, these should be calculated for each line apart based on its distance to the image plane and the pitch of the lens. These are huge and complicated tasks that are needed because the projection through the lenses is applied immediately on the geometric mesh, whereas, In *DIVGL* and *DCTarget* algorithms, a normal projection is applied on the geometric mesh by multiplying the object point coordinates by the projection matrix so that the object mesh triangles are projected and textured, and then the resulting image pixels are remapped so that the new values and locations of the pixels reflect the effect of the virtual lens array. As a result, the algorithms that are implemented in *FPPM* and *FSAM* methods is much more complicated than the algorithms that are implemented in both *DIVGL* and *DCTarget* methods, and therefore, the speed of rendering is lower in comparison to *DIVGL* and *DCTarget* methods.

- In *FPPM* and *FSAM* methods, the pinholes that are seen by each point of the scene object points must be calculated to determine the micro images each object point should be projected and mapped to. In *DIVGL* method, the part of the scene volume is calculated for each lenslet or microlens, the calculated part that is seen by the microlens or the pinhole is projected through the specified pinhole to form the micro image, then, the efforts that are paid in *DIVGL* method are almost equal to these in *FPPM* and *FSAM* methods to fulfil the same targets, and therefore, at this stage of processing, *FPPM*, *FSAM* and *DIVGL* methods are at the same level of complexity and speed of rendering. On contrary, in *DCTarget* method, this stage of processing is not required because the micro images that are formed by the virtual pinholes that are seen by that object points include the projections of the relevant object points. This procedure is implicitly implemented when the pixel values extracted from the projections and mapped to the correct locations in the micro images on the image plane. In *CGIP*, each object point is considered as the centre of a number of points forming the elemental image, in this case, each point should be projected through every lens in the lens array and that makes the process is even more complicated than in *FPPM* and *FSAM* because the calculation of the relevant pinholes or lenses reduces the speed of the process and increases the complexity and, so *CGIP* is marked with (×) against this step.
- The location of each pixel is calculated in *FPPM*, *FSAM* and *DCTarget* methods, whereas, it is not required in *DIVGL* method. In *DCTarget* method, the pixels are extracted and mapped to the new image, for this reason, the original location of the pixels are calculated, and then the new location in the image plane are calculated. This burden is not needed in *DIVGL* method in which the micro images are mapped as they are to their specified locations and as a result, the pixels are implicitly mapped to their correct locations without the need for calculating these locations. The pixel locations are calculated in *FPPM* and *FSAM* methods when calculating the locations where the rays hit the image plane. In *CGIP*, for each an object point, a set of elemental image points are plotted and therefore, the location of the pixels carrying the elemental image point intensity values should be calculated.
- *FPPM* and *FSAM* methods use the integral ray tracing technique to carry out the projections, whereas, *DIVGL* and *DCTarget* methods uses the method of multiplying the object point coordinates by the projection matrix. The later method requires less calculations and processing than the former one, and the complexity is proportional to

the amount of processes the computer needs to implement. Therefore, *DIVGL* and *DCTarget* are less complicated than that of *FPPM* and *FSAM* methods and the speed of rendering is higher.

- If the number of object points in a scene is specified, and each object point is projected a number of times during the image rendering process, we would calculate the number of projections that are needed to create the 3-D image of the same scene in each of the mentioned methods. The complexity is proportional to the number of projections. In *FPPM* and *FSAM* methods, the number of projections is the number of the original and calculated object points multiplied by the number of pinhole each object point sees. The total number of points is much higher than the original number of points, and average number of pinholes that are seen by the points is multiplied by the total number of points. In *DIVGL* method, only the original points are projected, however, each point is projected a number of times equals to the number of pinholes (or lenses) that are seen by this object point because the scene is divided to a number of small frustums equals to the number of lenses or pinholes. Then the number of projections in *DIVGL* is much lower than that of *FPPM* and *FSAM* methods and lower than that of *DCTarget* method because the number of projections in *DCTarget* is the number of original object points multiplied by the number of projections. The number of projections equals to the number of pixels under each microlens. In general, based on logical estimation and experience, the average number of lenses or pinholes that are seen by the points in a scene is less than the number of pixels under each microlens. In *CGIP*, each object point produces a proportional number of elemental points to the depth. The number of points implies on an equal number of processes to produce such points. Taking into consideration that each object point is projected through each possible lens, and the higher depth requires more points, it is obvious that the number of processes is higher in *CGIP* than these in the other considered methods, complexity is higher and speed of rendering is lower, therefore, *CGIP* is marked as (×) against this step.
- In *FSAM* method, the surface of the lenses are split into equally spaced lens points to take into account the refraction the rays suffer when passing through the lenses. This is not required in *FPPM* as the refraction doesn't occur in the pinhole model. When rendering the image, the refraction of the rays are neglected in *DIVGL* and *DCTarget* algorithms, however, these can be taken into account by mapping the pixels so that the refraction effect is considered. Omitting this step reduces the complexity of image

generation on the expenses of the quality of the rendered 2-D content and as a result the quality of the 3-D image. This step is irrelevant in *CGIP*.

- In *DIVGL* and *DCTarget* algorithms, remapping the pixels is essential to produce images with the required characteristics by implementing necessary processes including the correction of the pseudoscopic image and split the 3-D image into two parts behind and in front of the display screen. In *FPPM* and *FSAM* methods, these steps are implemented as the pixel locations are calculated and the object points are projected to the calculated pixel location and these pixels are not remapped again, however, the quality of images is affected and other complexities should be added to the rendering process to produce images with characteristics similar to these of the images produced in *DIVGL* and *DCTarget* algorithms. In *CGIP*, the calculated elemental images are made orthoscopic and therefore, the pixels are remapped.
- The shading function is less complicated in *DCTarget* (OpenGL) than it is in *FPPM* and *FSAM* methods that use interpolative shading. The interpolation of intensity values of the 3 corners of the transferred triangles at the image plane needs to check the orientation of the corners. In *FSAM* method, the whole triangle side is projected on the image plane through the lens array so the line is discontinuous. A further technique to find the orientation of each pixel used in making up a line is therefore necessary [8]. In *FPPM* the corners are projected through a lens array and need to interpolate their intensities after projection and calculate each pixel intensity value within the triangle taking into account the effect of the lenses, the effects of the lights, the effect of the ambient light, the depth values etc. These requirements need extra algorithms and higher number of calculations in comparison with these that are needed in *DIVGL* or *DCTarget* algorithms in which OpenGL is used to implement the shading only one time in the original scene then the scene is projected through the lens array in *DIVGL* and projected orthogonally in *DCTarget* without the need for calculating the pixel intensities again as these calculations are already done and the pixel intensity values are calculated smoothly and normally in advance taking into account all the effects. Therefore, there is no need for applying extra algorithms to implement the required shading on the image plane and calculate the pixel intensity values. In order to reduce the processing time in *FPPM* and *FSAM* methods, drawing a bounding box around each triangle was introduced to save searching time, however, searching the edges of the bounding boxes needs extra processing time and therefore,

higher complexity is needed. Information about the shading in *CGIP* are not available, so the complexity added in relation to this step is not counted.

Table 6.4 shows a comparison between few algorithms devoted to create 3-D autostereoscopic integral images on the basis of complexity and speed of rendering. In comparison to the method with higher complexity and lower speed of rendering, the method with a faster rendering and lower complexity for implementing the related procedure is marked with the symbol (✓), and the method with a lower speed of rendering and higher complexity in applying a procedure is marked with the symbol (×). From Table 6.3, the comparison between the discussed 5 methods shows that *DIVGL* or *DCTarget* algorithms are simpler and faster than the other methods in terms of the implementation steps.

	CGIP	**FPPM**	**FSAM**	**DIVGL**	**DCTarget**
Correct pseudoscopic image	×	×	×	✓	✓
Draw lines between object points	✓	×	×	✓	✓
Calculate pinholes for object points	×	×	×	×	✓
Calculate the location of pixels	×	×	×	✓	×
Ray tracing	✓	×	×	✓	✓
Number of object point projections	×	×	×	✓	✓
Split the surface of the lenses	✓	✓	×	✓	✓
Remapping the pixels	×	✓	✓	×	×
Shading after projection	✓	×	×	✓	✓

Table 6.4: A comparison between different methods on the basis of complexity.

Chapter 7
Conclusions and Further Work

7.1 Conclusions

With the development of this research from the initial theoretical ideas to the final implementation of the static and animated integral images, new techniques and tools were introduced to convert computer-generated images to 3-D autostereoscopic integral images that can be viewed without special glasses using the Application Interface OpenGL. In this chapter, some specific and general conclusions of the work are stated.

Integral imaging is the technique that provides 3-D autostereoscopic images viewable without the need for special glasses. Research and literature about integral imaging is limited and ongoing. An integral image quality is highly dependent on the parameters and characteristics of the devices employed in the imaging system. Attempts to produce such a system (i.e. 3-D cameras) are taking place. It would be useful to simulate the system and tune the parameters of the devices to optimise the performance of the system, and as a result, the quality of the produced integral images. In this thesis, a software tool based on new algorithms has been designed and implemented to simulate an integral imaging system in which the parameters can be adjusted to optimise the image quality, the computational and time costs, and the materials consumption. The algorithms used in the tool were introduced to provide fast and effective rendering of integral images. In comparison with other methods aimed to generate integral images, the introduced methods and tools in this study were more convenient for faster-to-produce, better quality and easier-to-tailor integral images with lower computational, time and material costs.

Most of the objectives of this study were met. Computer generated 3-D autostereoscopic static and animated integral images using original algorithms and application software were produced. The quality of the resultant images was higher than those images produced with other methods. These images were provided with 3-D effect and viewable with naked eyes (i.e. without using special glasses as it is the case for stereoscopic 3-D images). The size of the collected and processed data needed to produce 3-D integral images was reduced. A software tool that allows user to tune the system features and characteristics and covert computer-generated 2-D scenes to 3-D integral images has been designed and implemented. The software tool employs the API OpenGL and libraries based on the portable OpenGL; therefore, the software is portable between different operating systems and able to deal with various image formats. The software is special application software to simulate the 3-D integral imaging systems and help the designer to select the parameters of the system devices and generate the required 3-D images based on the selected parameters. Good quality 3-D integral images with fast and effective computations were achieved.

7.1.1 Findings

- Using the introduced algorithm Dividing Image Volume using OpenGL (*DIVGL*), good quality integral images were produced and subjectively evaluated as good quality integral images. Higher quality than images produced with an external method was proved by the subjective assessment.
- Displacing the Camera Target (*DCTarget*) algorithm was a developed method based on *DIVGL* and better quality computer-generated 3-D autostereoscopic integral images with a faster and less complicated process were produced with this method. The results of the subjective evaluation proved that the quality of these images exceeded the quality of those produced by *DIVGL*.
- The software designed and used to implement the above mentioned algorithms and produce the required integral images was well appreciated and accepted by users.
- The method introduced to evaluate the quality of the integral images based on the subjective evaluation of each one of the factors that are affecting the image quality of an integral image was used and significant results were acquired from that quality evaluation.

In Chapter 3, in order to compare the results of the approaches introduced in this study with the results of the closest technique, a previously suggested technique based on projecting the scene in the capture stage using projection points has been developed and implemented. The technique was implemented with OpenGL in the environment of C++. In one approach, the resulting images were successful and comparable to the images produced with the new techniques; however, other approaches were not satisfactory. The other part of Chapter 3 was devoted to explain the method of dividing the view volume to capture, project a scene introduce, describe and implement the algorithm *DIVGL* that is based on the view volume dividing method using OpenGL. In order to simulate the real life imaging with lens array, each microlens or cylindrical lens array is supposed to image the whole scene and produce a micro image of the whole scene to be recorded on the image plane, and then the micro images are displayed on a display screen. However, due to the limitation in the resolution of the display devices, a micro image of the whole scene is unable to be accommodated in the specified area on the display screen. The View Volume Dividing method allows each lenslet to capture an approximated viewing frustum of the complete viewing frustum that is supposed to be imaged by each lenslet in the ideal case. The left, right, upper and lower clipping planes of the frustum determine the viewing volume for each lenslet or microlens and therefore the micro-image under each microlens of that particular viewing frustum.

In Chapter 4, the *DCTarget* algorithm was introduced, explained and implemented with OpenGL. High quality static and animated integral images were rendered. The images were produced to be viewed using either cylindrical lens array or spherical lens array. *DCTarget* method is based on the principal of *DIVGL* method. *DCTarget* method aimed to extract the perspective projections of a scene from its orthogonal projections. The idea is about finding the pixel values of the perspective projection of a scene by implementing the orthogonal projections of that scene and extract specific pixels from each projection to build the required perspective projection. The number of orthogonal projection is equal to the number of pixels in each direction under each lens, whereas, the number of perspective projections needed to render the same cluster of micro images must be equal to the number of lenses in the lens array. Usually, the number of lenses in the lens array is much higher than the number of pixels under the micro lens in any direction. For example, a cylindrical lens array with *128* lenslets, each lenslet covers *8* pixels in the horizontal direction. As a result, the rendering process is much easier, faster, time effective, and lower computation costs.

DCTarget method is achieved by picking up specific pixels from each image of a collection of orthographically projected images with selected rotation angles of the camera targets and mapping the selected pixels to the final image. It is possible to display objects so that they are located in front or behind the screen by mapping the selected pixels in two different ways. The pixels that are belonging to the part of the scene located in front of the screen are mapped in a different way to these that are belonging to the part located behind the screen.

The orthographic projections simulate the perspective projection of the scene with the same lenses array. The projection of the points on the image plane is equivalent to the perspective projection with the same microlens array. The perspective characteristics are available in the integral image. The number of micro images that represent an object point expresses its distance to the screen. The number of lenses representing a specific object point encodes the distance of the composed real object point on the display screen. If the number of perspective projections is equal to the number of lenticular lenses or spherical lenses, and the number of orthogonal projections is equal to the number of pixels under each microlens or the number of columns of pixels under each lenticular lenslet, then the number of orthogonal projections is lower than the number of perspective projections. In other words, implementing a number of orthographic projections that is equal to the number of rotation angles is simpler than implementing a number of perspective projections that is equal to the number of lenslets. Therefore, a large amount of time and computations can be saved.

In Chapter 5, portable application software devoted to implement the mentioned algorithms has been introduced, explained and implemented. Examples of the integral images produced by the tool were shown.

With the various facilities provided by the application software, including the friendly user human-machine interface, the user is able to select options related to the scene, the lens array, and the required features of the resulting images. These options include the required characteristics of the lens array, the type of replay, the location of the objects in respect to the display screen, the 2-D static or animation scene to be converted to 3-D autostereoscopic scene or animation.

The suggested algorithms and the combined application software that is employed to implement these original algorithms provide a flexible, fast and effective technique to render 3-D autostereoscopic integral images and videos with high quality to be viewed by viewers with naked eyes.

The quality, complexity, and speed evaluation process was explained, the results were analysed, and the quality, speed and complexity of the rendered images were evaluated.

7.1.2 Limitations

Objective quality evaluation of the generated integral images with *DIVGL* and *DCTarget* methods against other methods requires the actual images generated with other methods and the data combined such as the depth map. The images should be belonging to the same category of the images generated with *DIVGL* and *DCTarget* methods. The category and the image type should be computer-generated 3-D autostereoscopic integral images. Despite that the images and materials from different categories such as stereoscopic images are easily available and accessible via databases; the materials needed for the comparison between different methods generating the required type of images are not easily accessible. Therefore, only subjective quality evaluation has been implemented in this thesis. Objective evaluation was implemented on the basis of the results of the subjective quality evaluation. *DIVGL* and *DCTarget* methods were proved to be better quality than that of a selected method that produces images from the same category.

Speed and complexity were analytically evaluated and compared with other methods based on the theoretical analysis of the rendering process. Practical and real rendering of other methods was not available and therefore, the advantages of the introduced methods over other methods were proved theoretically but not practically. *DIVGL* and *DCTarget* methods were proved to be faster and less complex that selected methods that produces the same type of images.

The 3-D integral images produced with the techniques introduced in this study are derived from computer-generated scenes and images, thus, these are not suitable for imaging real scenes. These techniques are suitable for producing animated integral images. The produced animations can be saved and replayed later. However, the techniques can be suitable for real-time integral imaging if the devices are fast enough to generate the integral image of the frame and display it within an acceptable time (e.g. in the video games). The scenes can be designed as normal 3-D scenes; the technique would be applied to the images and these would be displayed on a special screen. Generating the images and displaying them in real-time is not possible with the normal PCs due to the limitation in the image processing speed.

The size of the integral images implemented in this study is limited to the size of the PC display screen, however, with simple modifications, the application software that is based on the introduced algorithms allows user to render integral images with unlimited size as long as the devices are suitable for rendering such images. In addition, the resolution of the display screen was supposed to be the resolution of the PC screen that is proportionally low resolution. Higher resolution can be considered and better quality integral images can be implemented when suitable devices are available. On the other hand, the lack to a specialist display device adds a drawback to the quality because the thickness of the screen is not considered in the calculations.

Due to the approximations used in the calculations, the integral images that were rendered in the examples were not displayed perfectly using the lens array, and as a result, the quality was lower than it should be. Selecting the parameters correctly and accurately so that they are identical to the available lens array parameters enhances the quality of the 3-D autostereoscopic integral images. In Chapter 3, the capturing device was supposed to be a pinhole array, with more accurate calculations, the application software can be developed to calculate the integral images for a lens array capturing device.

7.2 Further Work

In order to reduce the computation processes in the capture stage of *DCTarget* method, the pixel values that are extracted from the orthographic projections can be calculated by projecting only the object points that are expected to form these pixel values. In this case, the discarded pixel values are not necessary to be calculated, and therefore, unnecessary calculations are avoided. These calculations can be avoided if the ray tracing technique is applied to the required pixels rather than all the pixels.

One technique can be investigated which is calculating the intersection points between the computer generated 3-D scene and planes (or lines) passing through the focal length of the lenses and hitting the image plane. If this method is applicable, a very fast and effective rendering can be achieved. More accuracy can be achieved if the pixels on the screen were distributed so that the micro-image shape is circular instead of being squared.

The characteristics of virtual lenses that were simulated in Chapter 3 and Chapter 4 were simplified. The light rays' intersection location with the image plane was approximated and some of the effect of the lenses on the ray were neglected. A higher accuracy can be achieved when taking into consideration all the lens characteristics and effects on the direction of the light. In this case, the pixel locations are accurately calculated and the quality of the resulting integral image is enhanced.

The application software can be developed to render images with higher width and length and print them out to be used for applications such as advertisements. Regarding the fact that the quality of the resulting images is dependent on the resolution of the display screen, printers provide higher resolution, and therefore, printing the images on transparent films and displaying the images with a normal light source provide 3-D images with high quality, which is a suitable way to use for different applications. In addition, another option can be added to the application software to implement the Dividing View Volume algorithm and apply *DIVGL* method on a selected scene in the same way *DCTarget* method is implemented.

The application software can be developed to be able to load several scenes and several textures at the same time. For each scene, the user should be able to set the parameters and the details of different objects in the scene.

Taking into consideration the advantages of OpenGL application interface, the software tool can be used with various platforms and devices such as mobile devices supplied with suitable display screens and 3-D autostereoscopic integral images can be instantly produced and displayed on these devices.

Appendix 1:

Viewer's vision and demographic data

Test Date/Time:

Observer Id:

Gender:	□ male □ female
Age:	□ under 20 □ 20-30 □ 30-40□ 40-50□ 50+
Educational level:	□ PhD □ Master □ Degree □ College □ School
Occupation:	□ Manager □ Researcher □ Employee □ Student □ Unemployed

How often do you watch 3D content?

□ once a week, □ once a month, □ several times a year, □ once a year, □ less often, □ never watched 3D content

Concerning your vision:

□ I don't wear glasses or contact lenses

□ I wear glasses or contact lenses to see clearly: □ at far □ at near

Experience of the viewer in 3-D imaging or image processing

□ I have good experience in 3-D imaging or image processing

□ I have moderate experience in 3-D imaging or image processing

□ I have some experience in 3-D imaging or image processing

□ I have no experience in 3-D imaging or image processing

Have a nice session!

Appendix 2:

Test Instructions

In this test, you will have to judge the quality of both the static and the animated 3-D autostereoscopic integral images.

- The images and videos will be displayed on this laptop mounted by a cylindrical lens array in front of you and the voting will be performed on the following pages.
- Three groups will be displayed and rated: groups 1 and group 2 each one includes 3 static images and one video, and group 3 includes 6 images.
- Group 1 will be displayed first followed by group 2 and then group 3.
- The video in each group will be displayed continuously for maximum of 5 minutes, whereas, each static image will be displayed once for 2 minutes maximum.
- Please stay focused during the test and feel free to move slightly left and right, backward and forward to judge the five criteria we explained in the training session.
- Two minutes are given after each display to fill in the form stating your opinion scores you worked out during the display. Write them next to the criterion you are scoring in the field specified for that image/video. However, the scores can be added during the display time.
- 5 minutes break will be given between the groups to finalize groups 1 and 2 scores.
- 5 minutes will be given to read these test instructions.
- 10 minutes at the end will be given to finalize the scores, fill in the demographic data form, and fill in the feedback form.
- The sequence of the test activities will be as follows: 5 minutes to read these instructions, 5 minutes video 1/group 1 display, 2 minutes break, 2 minutes static image 1/group 1 display, 2 minutes break, 2 minutes static image 2/group 1 display, 2 minutes break, 2 minutes static image 3/group 1 display, 5 minutes break, 5 minutes video 2/group 2 display, 2 minutes break, 2 minutes static image 1/group 2 display, 2 minutes break, 2 minutes static image 2/group 2 display, 2 minutes break, 2 minutes static image 3/group 2 display, the same is repeated for group 3, 10 minutes break

making the total of 83 minutes. However, the time can be much shorter if the decisions were taken faster.

- The three groups have 3 result tables.
- Each image/video of the three groups has a special column in the result tables. Group 1 and group 2 have the same result table, whereas, group 3 has a separated result table.
- Each criteria for the image/video has a special cell in the related image/video field.
- The criteria are the five criteria mentioned above: Comfort, Crosstalk, Parallax, Depth, and View Angle.
- Each cell will contain the Opinion Score of the quality of the corresponding image in terms of the related criteria.

The score can be any integer number between 0 and 10, where 0 indicates (bad), and 10 indicates (excellent).

Table of evaluation results

Quality evaluation of 3-D autostereoscopic static and video integral images for Group 1 and Group 2

Table of evaluation results

CRITEREA	**GROUP 1**				**GROUP 2**			
	Video	**Static Images**			**Video**	**Static Images**		
	Rotating cube	***Axe man 1***	***Cutlery***	***Globe***	***Angel in the castle***	***Boy***	***Angel and warriors***	***Axe man 2***
Comfort								
Crosstalk								
Parallax								
Depth								
View Angle								

Quality evaluation of 3-D autostereoscopic static and video integral images for Group 3

Table of evaluation results

CRITEREA	GROUP 3					
	Video	Static Images				
	Actor	*Ghost*	*Pots*	*Network*	*Ballerina*	*3-Tee*
Comfort						
Crosstalk						
Parallax						
Depth						
View Angle						

Appendix 3:

Assessment Feedback

1) What is your opinion in 3-D technology and 3-D imaging?

□ Very important(4) □ Important(3) □ Reasonable(2) □ Less important(1) □ Unimportant(0)

2) What is your opinion in 3-D autostereoscopic imaging?

□ Very important(4) □ Important(3) □ Reasonable(2) □ Less important(1) □ Unimportant(0)

3) What is your opinion in 3-D integral imaging and its characteristics?

□ Very important(4) □ Important(3) □ Reasonable(2) □ Less important(1) □ Unimportant(0)

4) What is your opinion in the application software that is used to produce our images?

□ Very good (4) □ good (3) □ acceptable (2) □ Less acceptable (1) □ Unacceptable (0)

5) What is your opinion in the introduced methods of generating integral images?

□ Very good (4) □ good (3) □ acceptable (2) □ Less acceptable (1) □ Unacceptable (0)

6) What is your opinion in our assessment in which you have participated?

□ Very simple (4) □ Simple (3) □ Average (2) □ Difficult (1) □ Very difficult (0)

Thank you very much for taking part in this assessment!

Appendix 4:

The subjective assessment results and calculations

The viewer's opinions of the images group 1:

Viewer number

1

8	7	9	9	0.825
8	6	7	6	0.675
7	6	5	7	0.625
7	8	7	8	0.75
8	7	6	6	0.675
0.761577	0.684105	0.69282	0.729383	0.7134 42

2

8	9	8	9	0.85
6	7	7	7	0.675
8	7	7	8	0.75
8	7	8	7	0.75
8	7	8	6	0.725
0.764199	0.744312	0.761577	0.746994	0.7521 64

3

9	7	8	8	0.8
7	6	7	8	0.7
6	7	6	7	0.65

7	6	5	8	0.65
8	7	7	6	0.7
0.746994	0.661816	0.667832	0.744312	0.70214
4				
7	5	5	7	0.6
5	6	5	4	0.5
6	5	4	6	0.525
6	5	6	6	0.575
5	6	4	5	0.5
0.584808	0.542218	0.485798	0.56921	0.541526
5				
8	7	6	5	0.65
6	5	5	6	0.55
6	5	6	7	0.6
6	7	6	7	0.65
5	6	5	5	0.525
0.627694	0.60663	0.562139	0.60663	0.597181
6				
10	9	8	9	0.9
7	7	7	7	0.7
8	7	6	7	0.7
9	8	7	7	0.775
8	7	8	9	0.8
0.846168	0.764199	0.723878	0.78613	0.77854
7				

6	5	4	6	0.525
4	5	4	5	0.45
5	6	5	6	0.55
5	6	5	5	0.525
4	5	6	4	0.475
0.485798	0.542218	0.485798	0.525357	0.506335
8				
8	7	7	8	0.75
7	6	7	7	0.675
6	6	6	5	0.575
7	6	6	6	0.625
8	7	6	7	0.7
0.723878	0.641872	0.641872	0.667832	0.667739
9				
5	4	4	5	0.45
4	3	5	3	0.375
3	4	4	5	0.4
5	4	5	5	0.475
5	3	4	5	0.425
0.447214	0.363318	0.442719	0.466905	0.426468
10				
7	8	6	7	0.7
6	5	7	5	0.575
6	7	5	7	0.625
7	7	6	8	0.7

7	6	7	5	0.625
0.661816	0.667832	0.6245	0.651153	0.6468 19
11				
8	7	6	6	0.675
7	6	5	7	0.625
6	6	7	7	0.65
6	8	5	5	0.6
7	7	5	6	0.625
0.684105	0.684105	0.565685	0.6245	0.6355 12
12				
6	6	5	6	0.575
5	4	5	4	0.45
5	6	5	6	0.55
7	6	7	7	0.675
6	5	4	6	0.525
0.584808	0.545894	0.52915	0.588218	0.5597 99
13				
9	7	7	8	0.775
8	7	7	8	0.75
7	7	6	8	0.7
8	7	8	8	0.775
7	6	7	7	0.675
0.783582	0.681175	0.702851	0.781025	0.7361 22
14				
7	8	8	7	0.75

5	4	4	6	0.475
4	5	4	4	0.425
4	5	4	5	0.45
3	4	3	2	0.3
0.479583	0.54037	0.491935	0.509902	0.502245
15				
8	7	8	7	0.75
4	3	3	4	0.35
3	3	3	4	0.325
6	5	6	6	0.575
4	3	4	3	0.35
0.531037	0.449444	0.517687	0.501996	0.498748
16				
8	7	9	8	0.8
3	2	3	4	0.3
3	3	4	4	0.35
4	3	3	3	0.325
4	4	5	3	0.4
0.477493	0.417133	0.52915	0.477493	0.47289
17				
7	6	6	7	0.65
5	5	6	4	0.5
8	7	6	8	0.725
7	6	6	8	0.675
6	5	5	6	0.55

0.667832	0.584808	0.581378	0.676757	0.6255
18				
8	9	9	8	0.85
8	9	8	8	0.825
8	7	7	6	0.7
10	8	8	9	0.875
8	6	7	8	0.725
0.843801	0.78867	0.783582	0.78613	0.7980 44
19				
9	9	9	10	0.925
7	8	7	8	0.75
9	7	8	9	0.825
9	8	9	9	0.875
7	8	7	8	0.75
0.825833	0.802496	0.804984	0.883176	0.8278 74
20				
8	7	7	8	0.75
4	3	4	4	0.375
7	5	5	5	0.55
5	4	6	6	0.525
5	5	5	5	0.5
0.598331	0.497996	0.549545	0.576194	0.5533 99
21				
10	8	7	9	0.85
7	5	5	7	0.6

5	6	7	7	0.625
7	5	5	7	0.6
6	6	5	5	0.55
0.719722	0.609918	0.588218	0.711337	0.653548
22				
5	5	5	5	0.5
7	7	7	7	0.7
5	5	5	5	0.5
8	8	8	8	0.8
5	5	5	5	0.5
0.613188	0.613188	0.613188	0.613188	0.613188
23				
9	8	8	7	0.8
9	8	7	7	0.775
5	5	5	5	0.5
5	6	5	6	0.55
6	6	5	5	0.55
0.704273	0.67082	0.613188	0.60663	0.647399
0.659293	0.613241	0.606934	0.644802	0.628549

The viewer opinions of the images group 2:

9	10	9	8	0.9
9	8	8	9	0.85
7	8	8	7	0.75
8	9	7	8	0.8
8	8	8	7	0.775
0.8234	0.8637	0.8024	0.7835	0.8167
08	13	96	82	77
9	9	9	8	0.875
10	7	7	8	0.8
9	9	8	7	0.825
9	10	8	9	0.9
9	8	7	9	0.825
0.9208	0.8660	0.7835	0.8234	0.8457
69	25	82	08	98
10	8	9	9	0.9
9	7	8	9	0.825
9	8	7	8	0.8
10	9	8	7	0.85
8	9	7	9	0.825
0.9230	0.8234	0.7835	0.8438	0.8406
38	08	82	01	84
7	6	7	8	0.7
7	7	8	6	0.7
7	6	5	7	0.625
8	8	7	7	0.75
7	6	6	6	0.625
0.7211	0.6648	0.6678	0.6841	0.6817
1	31	32	05	26
9	6	7	6	0.7
8	7	8	7	0.75
7	8	7	8	0.75
8	6	7	8	0.725
7	7	6	8	0.7
0.7835	0.6841	0.7028	0.7443	0.7253
82	05	51	12	45
10	10	9	10	0.975

8	9	7	8	0.8
9	10	9	8	0.9
10	9	8	9	0.9
9	10	8	9	0.9
0.923038	0.961249	0.823408	0.883176	0.89673
8	5	6	7	0.65
8	6	7	7	0.7
8	6	6	5	0.625
7	6	8	7	0.7
6	5	5	6	0.55
0.744312	0.562139	0.648074	0.644981	0.647399
9	8	10	9	0.9
8	7	8	7	0.75
7	8	7	7	0.725
8	7	7	8	0.75
9	8	7	8	0.8
0.823408	0.761577	0.78867	0.783582	0.78748
7	7	6	8	0.7
6	5	4	4	0.475
7	6	5	6	0.6
6	7	5	7	0.625
7	5	6	6	0.6
0.661816	0.60663	0.525357	0.634035	0.604359
9	8	8	8	0.825
8	7	7	6	0.7
7	8	7	8	0.75
9	8	6	8	0.775
8	7	6	8	0.725
0.823408	0.761577	0.684105	0.764199	0.756224
8	7	8	6	0.725
7	6	7	7	0.675
7	7	8	7	0.725
8	7	6	7	0.7
6	8	7	7	0.7
0.723878	0.702851	0.723878	0.681175	0.705248

8	9	6	8	0.775
7	7	6	5	0.625
8	7	6	7	0.7
8	8	7	6	0.725
8	6	7	7	0.7
0.7810 25	0.7469 94	0.6418 72	0.6678 32	0.7066 65
10	9	8	9	0.9
9	8	8	9	0.85
8	9	9	8	0.85
9	8	9	7	0.825
7	8	8	9	0.8
0.8660 25	0.8414 27	0.8414 27	0.8438 01	0.8456 51
9	8	7	8	0.8
7	6	8	5	0.65
6	5	5	7	0.575
7	7	6	5	0.625
5	4	5	4	0.45
0.6928 2	0.6164 41	0.6308 72	0.5983 31	0.6302 78
9	8	8	7	0.8
7	6	5	5	0.575
6	5	4	5	0.5
7	5	7	7	0.65
6	4	5	4	0.475
0.7085 2	0.5761 94	0.5983 31	0.5727 13	0.6113 51
8	8	10	9	0.875
5	6	5	4	0.5
5	5	6	5	0.525
5	4	5	4	0.45
6	4	5	6	0.525
0.5916 08	0.5603 57	0.6496 15	0.5899 15	0.5948 74
8	8	7	7	0.75
7	6	8	6	0.675
9	8	8	9	0.85
8	8	7	8	0.775
7	7	5	7	0.65
0.7835 82	0.7443 12	0.7085 2	0.7469 94	0.7434 72

10	10	9	10	0.975
10	8	8	9	0.875
9	9	8	8	0.85
10	9	9	9	0.925
9	8	8	9	0.85
0.9612 49	0.8831 76	0.8414 27	0.9022 19	0.8963 12
10	10	10	9	0.975
9	8	8	7	0.8
10	9	9	8	0.9
10	9	10	9	0.95
9	9	8	7	0.825
0.9612 49	0.9022 19	0.9044 34	0.8049 84	0.8926 09
7	7	6	5	0.625
7	7	7	6	0.675
7	8	9	8	0.8
7	7	8	6	0.7
6	7	6	6	0.625
0.6811 75	0.7211 1	0.7293 83	0.6276 94	0.6880 23
10	10	9	9	0.95
8	7	9	8	0.8
8	7	8	8	0.775
7	8	8	6	0.725
7	6	6	7	0.65
0.8074 65	0.7720 1	0.8074 65	0.7668 12	0.7862 89
7	7	7	7	0.7
9	9	9	9	0.9
7	7	7	7	0.7
8	8	8	8	0.8
7	7	7	7	0.7
0.7641 99	0.7641 99	0.7641 99	0.7641 99	0.7641 99
10	10	9	9	0.95
10	9	8	7	0.85
7	8	6	7	0.7
7	7	6	6	0.65
7	8	8	7	0.75
0.8330	0.8461	0.7496	0.7266	0.7874

67	68	67	36	01
0.7958 2	0.7492 48	0.7304 8	0.7340 21	0.7502 13

Viewer’s opinion scores group 3:

6	6	7	7	7	6	0.65	0.65
7	5	4	8	7	7	0.725	0.633333
5	3	2	5	4	6	0.5	0.416667
6	4	4	6	4	5	0.525	0.483333
5	4	3	6	4	5	0.5	0.45
0.584808	0.451664	0.43359	0.648074	0.54037	0.584808	0.587154	0.535413
6	6	7	8	7	9	0.75	0.716667
7	5	6	7	5	6	0.625	0.6
6	5	5	9	5	6	0.65	0.6
7	3	3	6	7	8	0.7	0.566667
7	4	3	6	5	7	0.625	0.533333
0.661816	0.471169	0.505964	0.729383	0.588218	0.729383	0.671751	0.606493
8	4	5	7	7	8	0.75	0.65
7	3	4	8	6	8	0.725	0.6
6	4	3	8	5	7	0.65	0.55
7	3	3	5	5	7	0.6	0.5
7	4	5	6	7	6	0.65	0.583333
0.702851	0.363318	0.409878	0.689928	0.60663	0.723878	0.677219	0.57884
4	2	1	4	5	6	0.475	0.366667
3	3	3	5	6	6	0.5	0.433333
5	2	1	4	3	4	0.4	0.316667
3	2	3	5	6	5	0.475	0.4
4	2	3	4	6	5	0.475	0.4
0.387298	0.223607	0.240832	0.442719	0.532917	0.525357	0.466235	0.385357
4	4	3	5	5	6	0.5	0.45
5	4	3	4	5	7	0.525	0.466667
5	4	4	6	7	6	0.6	0.533333
5	5	4	5	6	6	0.55	0.516667
4	4	3	3	5	5	0.425	0.4
0.462601	0.4219	0.343511	0.471169	0.565685	0.603324	0.523211	0.475745
7	6	6	7	6	8	0.7	0.666667
5	4	4	6	7	6	0.6	0.533333

6	3	2	5	6	5	0.55	0.45
7	3	2	5	5	5	0.55	0.45
6	4	3	6	5	7	0.6	0.516667
0.6245	0.414729	0.371484	0.584808	0.584808	0.630872	0.602495	0.529308
3	3	2	5	4	5	0.425	0.366667
4	3	2	4	5	6	0.475	0.4
4	3	2	4	5	4	0.425	0.366667
4	3	2	4	5	6	0.475	0.4
3	2	1	5	4	5	0.425	0.333333
0.363318	0.282843	0.184391	0.442719	0.462601	0.525357	0.445674	0.374166
5	4	3	7	6	6	0.6	0.516667
7	4	4	8	5	6	0.65	0.566667
5	4	3	5	6	5	0.525	0.466667
6	4	5	6	5	5	0.55	0.516667
7	4	4	5	6	7	0.625	0.55
0.60663	0.4	0.387298	0.630872	0.562139	0.584808	0.591819	0.524457
4	1	1	4	4	5	0.425	0.316667
3	1	1	5	3	4	0.375	0.283333
4	1	1	4	3	4	0.375	0.283333
3	1	1	4	3	3	0.325	0.25
4	2	1	4	3	3	0.85	0.283333
0.363318	0.126491	0.1	0.4219	0.32249	0.387298	0.507937	0.284117
5	1	3	6	7	6	0.6	0.466667
5	1	2	5	4	6	0.5	0.383333
4	1	2	6	4	6	0.5	0.383333
3	1	1	6	5	4	0.45	0.333333
4	2	1	6	5	5	0.5	0.383333
0.426615	0.126491	0.194936	0.581378	0.511859	0.545894	0.512348	0.392358
4	1	2	5	6	5	0.5	0.383333
6	1	1	4	5	5	0.5	0.366667
5	3	2	5	6	5	0.525	0.433333
4	1	2	5	4	4	0.425	0.333333
3	2	1	5	4	4	0.4	0.316667
0.451664	0.178885	0.167332	0.481664	0.507937	0.462601	0.472493	0.368932
4	2	3	5	5	6	0.5	0.416667
4	1	2	5	4	5	0.45	0.35
5	1	2	5	5	4	0.475	0.366667
4	1	1	5	4	4	0.425	0.316667
5	2	1	5	4	4	0.45	0.35
0.442719	0.148324	0.194936	0.5	0.442719	0.466905	0.460706	0.361478

7	1	1	7	6	5	0.625	0.45
6	5	4	7	5	5	0.575	0.533333
5	1	1	7	6	6	0.6	0.433333
6	2	2	6	5	5	0.55	0.433333
6	4	5	7	5	5	0.575	0.533333
0.603324	0.306594	0.306594	0.681175	0.542218	0.521536	0.585555	0.478946
7	5	5	7	8	7	0.725	0.65
4	1	2	4	4	3	0.375	0.3
3	1	1	4	3	3	0.325	0.25
3	1	1	3	2	2	0.25	0.2
2	1	1	3	2	2	0.225	0.183333
0.417133	0.240832	0.252982	0.444972	0.440454	0.387298	0.420714	0.36017
6	2	3	6	5	5	0.55	0.45
4	1	2	4	3	3	0.35	0.283333
2	1	1	4	2	2	0.25	0.2
5	1	2	5	5	4	0.475	0.366667
3	1	1	3	2	3	0.275	0.216667
0.424264	0.126491	0.194936	0.451664	0.36606	0.354965	0.397178	0.317543
9	7	8	6	6	5	0.65	0.683333
2	1	1	3	2	2	0.225	0.183333
3	1	1	2	1	1	0.175	0.15
2	1	1	3	2	2	0.225	0.183333
3	0	1	3	2	2	0.25	0.183333
0.462601	0.32249	0.368782	0.36606	0.31305	0.275681	0.351248	0.343592
5	4	3	5	5	4	0.475	0.433333
5	3	2	4	5	5	0.475	0.4
4	1	2	6	4	6	0.5	0.383333
5	3	3	7	5	3	0.5	0.433333
5	1	2	5	4	4	0.45	0.35
0.481664	0.268328	0.244949	0.549545	0.462601	0.451664	0.480364	0.401248
8	5	6	7	6	6	0.675	0.633333
7	4	4	7	6	5	0.625	0.55
7	6	6	8	7	7	0.725	0.683333
8	5	4	7	7	8	0.75	0.633333
8	6	5	8	7	7	0.75	0.683333
0.761577	0.525357	0.507937	0.74162	0.661816	0.667832	0.706665	0.638531
8	5	6	9	8	10	0.875	0.766667
9	6	7	8	7	7	0.775	0.733333
7	6	5	8	7	8	0.75	0.683333

9	4	5	7	6	9	0.775	0.666667
7	3	4	7	6	8	0.7	0.583333
0.804984	0.493964	0.549545	0.783582	0.684105	0.846168	0.777094	0.689525
4	3	4	6	7	5	0.55	0.483333
5	2	2	4	4	5	0.45	0.366667
5	3	4	5	4	5	0.475	0.433333
3	1	1	4	3	3	0.325	0.25
4	2	2	4	4	5	0.425	0.35
0.426615	0.232379	0.286356	0.466905	0.460435	0.466905	0.450971	0.384924
8	7	7	8	8	7	0.775	0.75
5	3	4	5	4	5	0.475	0.433333
5	4	4	5	4	5	0.475	0.45
4	2	2	5	4	5	0.45	0.366667
5	3	2	5	4	4	0.45	0.383333
0.556776	0.417133	0.4219	0.572713	0.505964	0.52915	0.539792	0.496823
3	3	3	3	3	3	0.3	0.3
5	5	5	5	5	5	0.5	0.5
2	2	2	2	2	2	0.2	0.2
5	5	5	5	5	5	0.5	0.5
4	4	4	4	4	4	0.4	0.4
0.397492	0.397492	0.397492	0.397492	0.397492	0.397492	0.397492	0.397492
8	8	7	8	9	9	0.85	0.816667
6	5	6	7	6	5	0.6	0.583333
4	3	2	4	5	5	0.45	0.383333
4	3	2	4	4	3	0.375	0.333333
5	3	3	4	4	5	0.45	0.4
0.560357	0.481664	0.451664	0.56745	0.589915	0.574456	0.570636	0.533906

The three methods opinions RMS:

Data of Figure (6.17)

viewer	DIVGL Method	DCTarget Method	OPT ISTM Method	ISTM Method
1	0.713442359	0.816777203	0.587154154	0.535412613
2	0.752163546	0.845798439	0.671751442	0.606492649
3	0.702139587	0.840684245	0.677218576	0.578839836
4	0.541525623	0.681725751	0.466234919	0.385356977
5	0.597180877	0.725344746	0.523211238	0.47574503

6	0.778540301	0.896730171	0.602494813	0.529307724
7	0.506334869	0.647398641	0.445673647	0.374165739
8	0.667738721	0.787480158	0.591819229	0.524457392
9	0.426468053	0.604359165	0.507937004	0.284116564
10	0.64681914	0.756224173	0.512347538	0.392357547
11	0.635511605	0.705248183	0.472493386	0.368932394
12	0.559799071	0.706664701	0.46070598	0.361478446
13	0.736121593	0.845650637	0.585555292	0.478945601
14	0.50224496	0.630277717	0.420713679	0.360169713
15	0.498748434	0.611350963	0.397177542	0.317542648
16	0.472890051	0.594873936	0.351247776	0.343592135
17	0.6254998	0.743471587	0.480364445	0.401248053
18	0.798044485	0.896311888	0.706664701	0.638531301
19	0.827873783	0.892608537	0.777093945	0.689524796
20	0.553398591	0.688022529	0.450971174	0.384924235
21	0.653548009	0.786288751	0.539791626	0.496823242
22	0.613188389	0.764198927	0.397492138	0.397492138
23	0.647398641	0.787400787	0.570635611	0.533905943

RMS of MOS and MOS of RMS:

Data of Figure (6.21)

	DIVGL_1	DCTarget_1	OPT ISTM_1	ISTM_1
1	0.713442359	0.816777203	0.587154154	0.535412613
2	0.752163546	0.845798439	0.671751442	0.606492649
3	0.702139587	0.840684245	0.677218576	0.578839836
4	0.541525623	0.681725751	0.466234919	0.385356977
5	0.597180877	0.725344746	0.523211238	0.47574503
6	0.778540301	0.896730171	0.602494813	0.529307724
7	0.506334869	0.647398641	0.445673647	0.374165739
8	0.667738721	0.787480158	0.591819229	0.524457392
9	0.426468053	0.604359165	0.507937004	0.284116564
10	0.64681914	0.756224173	0.512347538	0.392357547
11	0.635511605	0.705248183	0.472493386	0.368932394
12	0.559799071	0.706664701	0.46070598	0.361478446
13	0.736121593	0.845650637	0.585555292	0.478945601
14	0.50224496	0.630277717	0.420713679	0.360169713
15	0.498748434	0.611350963	0.397177542	0.317542648
16	0.472890051	0.594873936	0.351247776	0.343592135
17	0.6254998	0.743471587	0.480364445	0.401248053
18	0.798044485	0.896311888	0.706664701	0.638531301
19	0.827873783	0.892608537	0.777093945	0.689524796
20	0.553398591	0.688022529	0.450971174	0.384924235
21	0.653548009	0.786288751	0.539791626	0.496823242

22	0.613188389	0.764198927	0.397492138	0.397492138
23	0.647398641	0.787400787	0.570635611	0.533905943

ISTM_1	DIVGL_2	DCTarget_2	OPT ISTM_2	ISTM_2
0.535412613	0.716972	0.818299524	0.589514909	0.540552
0.606492649	0.75427	0.848470908	0.677199983	0.614322
0.578839836	0.705238	0.843457115	0.680821837	0.582747
0.385356977	0.545508	0.684469657	0.472072683	0.392122
0.47574503	0.600773	0.728712469	0.525694912	0.478032
0.529307724	0.780094	0.897717822	0.606246883	0.5352
0.374165739	0.509793	0.649876306	0.448498819	0.376871
0.524457392	0.668864	0.789309038	0.59611222	0.528625
0.284116564	0.430039	0.60695936	0.373751787	0.286916
0.392357547	0.651325	0.758322262	0.516436343	0.397862
0.368932394	0.639599	0.707945919	0.47596643	0.375014
0.361478446	0.562017	0.70943088	0.463085612	0.365934
0.478945601	0.737158	0.848170261	0.58706336	0.493574
0.360169713	0.505448	0.634616287	0.422464407	0.363945
0.317542648	0.500041	0.613939462	0.399238138	0.31973
0.343592135	0.475318	0.597873882	0.354347984	0.351444
0.401248053	0.627694	0.745851711	0.486368495	0.409792
0.638531301	0.800545	0.89701803	0.708211289	0.644357
0.689524796	0.829122	0.893221667	0.77970977	0.693725
0.384924235	0.555517	0.689840801	0.45521464	0.389932
0.496823242	0.657299	0.788438071	0.541150992	0.500606
0.397492138	0.613188	0.764198927	0.397492138	0.397492
0.533905943	0.648728	0.788884283	0.573044745	0.537584

Criteria MOS of each image:

Data of Figure (6.19)

	Cube Rotate	Axeman 1	Cutlery	Globe	Angel	Boy	Worriers	Axeman 2
Comfort	0.773913043	0.7043478	0.691304	0.734783	0.873913	0.817391	0.8	0.8
Crosstalk	0.604347826	0.5521739	0.573913	0.591304	0.795652	0.708696	0.773913	0.6869565
Parallax	0.591304348	0.573913	0.547826	0.621739	0.756522	0.743478	0.704348	0.7173913
Depth	0.665217391	0.6217391	0.613043	0.669565	0.8	0.756522	0.726087	0.7217391
ViewAngle	0.608695652	0.5695652	0.556522	0.552174	0.730435	0.691304	0.656522	0.7086957

Actor	Ghost	Pots	Network	Ballerina	3-Tee
0.578261	0.391304	0.417391	0.617391	0.608696	0.617391
0.526087	0.773913	0.326087	0.552174	0.491304	0.530435
0.465217	0.273913	0.252174	0.526087	0.452174	0.486957

0.491304	0.256522	0.256522	0.513043	0.465217	0.482609
0.482609	0.278261	0.256522	0.495652	0.443478	0.486957

MOS of images:

Data of Figure (6.18)

Cube Rotate (DIVGL)	Axeman 1 (DIVGL)	Cutlery (DIVGL)	Globe (DIVGL)	Angel (DCTarget)	Boy (DCTarget)
0.659292787	0.613240775	0.606933804	0.644802258	0.795819597	0.749248468

Worriers (DCTarget)	Axeman 2 (DCTarget)	Actor (ISTM)	Ghost (ISTM)	Pots (ISTM)
0.730480334	0.734021104	0.520648942	0.322701959	0.32683868

Network (ISTM)	Ballerina (ISTM)	3-Tee (ISTM)
0.549903966	0.506629715	0.532332

Average of RMS of the 23 viewers:

Data of Figure (6.20)

	DIVGL	DCTarget	OPT ISTM	ISTM
1	0.716971547	0.818299524	0.589514909	0.540552149
2	0.754270494	0.848470908	0.677199983	0.614322186
3	0.705238434	0.843457115	0.680821837	0.582747237
4	0.545508405	0.684469657	0.472072683	0.39212157
5	0.600773259	0.728712469	0.525694912	0.478031899
6	0.780093741	0.897717822	0.606246883	0.535199979
7	0.509792829	0.649876306	0.448498819	0.37687148
8	0.66886382	0.789309038	0.59611222	0.528624536
9	0.430038803	0.60695936	0.373751787	0.286916376
10	0.651325156	0.758322262	0.516436343	0.397862061
11	0.639598934	0.707945919	0.47596643	0.37501386
12	0.562017333	0.70943088	0.463085612	0.365933717
13	0.73715832	0.848170261	0.58706336	0.493573638
14	0.505447576	0.634616287	0.422464407	0.363945289
15	0.500041001	0.613939462	0.399238138	0.319729924
16	0.475317561	0.597873882	0.354347984	0.351444004
17	0.627693655	0.745851711	0.486368495	0.409791852

18	0.800545493	0.89701803	0.708211289	0.644356531
19	0.829122394	0.893221667	0.77970977	0.693724651
20	0.555516669	0.689840801	0.45521464	0.38993233
21	0.657298653	0.788438071	0.541150992	0.50060625
22	0.613188389	0.764198927	0.397492138	0.397492138
23	0.648727873	0.788884283	0.573044745	0.537584392

MOS criteria of each method:

Data of Figure (6.23) and Data of Figure (6.22)

	DIVGL Method	DCTarget	Optimized ISTM Method	ISTM Method
Comfort	0.726086957	0.822826087	0.605434783	0.538405797
Crosstalk	0.580434783	0.730434783	0.525	0.455797101
Parallax	0.583695652	0.730434783	0.482608696	0.40942029
Depth	0.642391304	0.751086957	0.488043478	0.410144928
ViewAngle	0.57173913	0.69673913	0.498913043	0.407246377
Criteria				
RMS	0.623595965	0.747487758	0.521955571	0.447062691

Publications

- *Shafik Salih, Amar Aggoun, Maysam Abbod, "*A 3-D Auto-stereoscopic Integral Images Generating Tools*",* Proc. IPCV'14 - The 2014 International Conference on Image Processing, Computer Vision, and Pattern Recognition, *ISBN: 1-60132-280-1*, July 21-24, 2014, Las Vegas, USA

- *Shafik Salih, Amar Aggoun, Maysam Abbod, "*Computer generation and rendering of integral images by displacing the virtual camera target*",* (In the process of publishing in: *IEEE Transactions on Image processing*).

- *Shafik Salih, Amar Aggoun, Maysam Abbod, "*Computer generation and rendering of integral images with the method of dividing image volume using OpenGL*",* (In the process of publishing in: *IEEE Transactions on Image processing*).

- Vasilis Michopoulos, Sarogini Grace Pease, Axel Bindely, Shafik Salihy, Paul Conway, Andrew West, "Design and Implementation of High-accuracy Remote Energy and Data Monitoring System for Industrial Applications Using the (Internet of Things)", (In the process of publishing in: *IEEE Journal of Systems Engineering and Electronics*).

References

[1] A. Aggoun, “Pre-processing of integral images for 3-D displays*,” Journal of Display Technology,* Vol. 2, No. 4, Dec 2006.

[2] N. A. Dodgson, “Autostereoscopic 3D displays,” *IEEE Computer*, Vol. 38, No. 8, pp. 31–36, 2005.

[3] G. A. Thomas and R. F. Stevens, “Processing of images for 3D display, U.S. Patent 6 798 409 B2, Sep 28, 2004.

[4] M. Martínez-Corral, R. Martínez-Cuenca, G. Saavedra, “Integral imaging: auto-stereoscopic images of 3D scenes”, *Department of Optics, University of Valencia, Burjassot, Spain, SPIE Newsroom.* DOI, 10.1117/2.1200611.0425, 17 Nov 2006.

[5] D. Hearn, M. P. Baker, “Computer graphics with OpenGL”, 3rd ed., Published by Pearson Prentice Hall, USA, Sep, 22, 2003.

[6] C. V. Berkel, J. A. Clarke, Philips Research Laboratories, UK, “Characterisation and optimisation of 3D-LCD module design”, SPIE Digital Library Vol. 3012, 10 Jan 1997.

[7] H. Ujike, Shin-ichi Uehara, “Human factors of autostereoscopic displays for standardization”, Japanese Ergonomics National Committee, VIMS 2009, 11th Jun 2009.

[8] G. E. Milnthorpe, “Computer generation of integral images using interpolative shading techniques”, PhD thesis, School of Engineering and Manufacture, De Montfort University, Leicester, Nov 2003.

[9] J. Kassebaum, N. Bulusu, W. Feng, “Smart camera network localization using a 3D target” Portland State University, Sep 2009.

[10] N. Davies, M. McCormick, and Li Yang, “Three-dimensional imaging systems, a new development,” *Applied Optics*, Vol. 27, Issue 21, pp. 4520-4528, 1988.

[11] R. Olsson, Y. Xu, “An interactive ray-tracing based simulation environment for generating integral imaging video sequences,” Proc. SPIE 6016, *Three-Dimensional TV, Video, and Display IV*, 60160F, Nov 15, 2005; doi:10.1117/12.630733.

[12] N. Davies, M. McCormick, and Li Yang, "Three-dimensional imaging systems: a new development," *Applied Optics*, Vol. 27, Issue 21, pp. 4520-4528, 1988.

[13] E. B. Javidi and F. Okano, “*Three-Dimentional Television, video, and Display Technologies”*, Springer, 2002.

[14] G. Milnthorpe, M. McCormick, N. Davies, "Computer modelling of lens arrays for Integral Image rendering," *Imaging Technologies Group,* SERCentre, De-Montfort University, Leicester LE1 9BH UK, 2002.

[15] A. Sokolov, "Autostereoscopy and Integral Photography by Professor Lippmann's Method," *MGU, Moscow State Univ*. Press, 1911.

[16] Y. Igarashi, H. Murata, M. Ueda "3D Display System Using a Computer Generated Integral Photograph" *Japan J. Appl. Phys*. Vol. 17, No. 9, 1978.

[17] G. Milnthorpe, M. McCormick, A. Aggoun, N. Davies, M. Forman, "Computer generated content for 3D TV displays," De Montfort University, UK, 2003.

[18] G. Lippmann, "Epreuves reversibles donnant la sensation du relief", *j. phys.,* Vol. 7, pp 821-825, 1908.

[19] H. E. Ives, "Optical Properties of a Lippmann Lenticulated Sheet", *J. Opt. Soc*. Amer. Vol. A 21, pp. 171-176, 1931.

[20] N. A. Valyus, "Stereoscopy", *Focal Press*, London, UK 1966.

[21] T. Okoshi, "Three-Dimensional Imaging Techniques", *Academic Press*, London, UK 1976.

[22] J. R. Moor, A. R. L. Travis, S. R. Lang and O. M. Castle, "The Implementation of a Multi-View Autostereoscopic Display", *IEE Colloq*. Stereoscopic Television 1992/173, pp. 4/5-4/16, 1992.

[23] N. Davies and M. McCormick, M. Brewin, "Design and analysis of an image transfer system using microlens arrays", *Opt. Eng*. Vol. 33, No. 11, pp. 3624-3633, Nov 1994.

[24] R.F. Stevens, N. Davies, G. Milnthorpe, "Lens arrays and optical system for orthoscopic three-dimensional imaging", *The Imaging Science Journal*, Vol. 49 pp. 151-164, 2001.

[25] T. Naemura, T. Yoshida and H. Harashima,"3-D computer graphics based on integral photography", *OPTICS EXPRESS 255,* Vol. 8, No. 2, 12 Feb 2001.

[26] F. Okano, H. Hoshino, J. Arai and I. Yuyama, "Real-time pickup method for a three dimensional image based on integral photography", *App. Opt.* Vol. 36, pp. 1598-1603, 1997

[27] J. Arai, F. Okano, H. Hoshino, and I. Yuyama, "Gradient-index lens-array method based on real-time integral photography for three-dimensional images", *App. Opt*. Vol. 37, pp. 2034-2045, 1998

[28] A. Alatan et al, "Scene representation technologies for 3DTV—a survey", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 17, No. 11, Nov 2007.

[29] Y. Jeong, S. Jung, J. Park, and B. Lee, "Reflection-type integral imaging scheme for displaying three-dimensional images", *Optics Letters* / Vol. 27, No. 9, May 1, 2002.

[30] E.H. Adelson, and J. Wang, “Single lens stereo with a plenoptic camera”, *IEEE Transaction on Pattern Analysis and machine Intelligence,* Vol. 14, No. 2, Feb 1992.

[31] A. Stern and B. Javidi, “3-D computational synthetic aperture integral imaging (COMPSAII)”, *Optics Express* 2446/22, Vol. 11, No. 19, Sep 2003.

[32] G. Lippmann, “La photographic intergrale” *C. R. Acad. Sci*. Vol. 146, pp. 446-451 (1908)

[33] J. S. Jang and B. Javidi, “Three-dimensional synthetic aperture integral imaging” *Opt. Lett*. Vol. 27, pp. 1144-1146, 2002.

[34] H. Hoshino, F. Okano, H. Isono and I. Yuyama,” Analysis of resolution limitation of integral photography” *J. Opt. Soc. Am*. Vol. 15, pp. 2059-2065, 1998.

[35] H. Choi, S. W. Okano, J. Kim, and B. Lee, ” A thin 3D-2D convertible integral imaging system using a pinhole array on a polarizer”, *Optics Express,* Vol. 14, No. 12, p. 5183, 12 Jun 2006.

[36] K. Choi, J. Kim, Y. Lim, and B. Lee, ” Full parallax viewing-angle enhanced computer generated holographic 3D display system using integral lens array” *Optics Express*, Vol. 13, No. 26, p. 10494, 26 Dec 2005.

[37] J.-H. Park, H.R. Kim, Y. Kim, J. Kim, J. Hong, S.D. Lee, and B. Lee, "Depth-enhanced three dimensional- two-dimensional convertible display based on modified integral imaging," *Opt. Lett*. Vol. 29, pp. 2734-2736, 2004.

[38] T. Motoki, H. Isono, I. Yuyama, “Present Status of Three-Dimensional Television Research”, *Proceedings of the IEEE*. Vol. 83, pp. 1009-1021, 1995

[39] G. Lippmann, “Epreuves reversibles”, *Comptes rendus hebdomadaires des Seances de l'Academie des Sciences,* Vol.-146, pp. 446-451, Mar 1908.

[40] C. B. Burkhardt and E. T. Doherty, “Beaded plate recording of integral photographs”, *ppl. Opt*. Vol. 8, No. 11, pp. 2329-2331, 1969.

[41] R. L. Demontebello, “Wide angle integral photography - the integram technique”, *Proc. SPIE*, Vol. 120, pp. 73-91, 1970.

[42] Y. A. Dudnikov, B. K. Rozhkov and E. N. Antipova, “Obtaining a Portrait of a Person by the Integral Photography Method”, *Sov. J. Opt*. Tech. Vol. 47, No. 9, pp. 562-563, 1980.

[43] M. Price, G.A. Thomas, “3D virtual production and delivery using MPEG-4”, *Proc. of the Int. Broadcasting Convention* (IBC 2000), Amsterdam, 8-12 Sep 2000.

[44] M. Born and E. Wolf, "Principles of optics: electromagnetic theory of propagation interference and diffraction of light" *4th ed. Pergamon Press*, Bath, 1970.

[45] T. Ito and K. Okano, "Color electro-holography by three colored reference lights simultaneously incident upon one hologram panel," *Opt. Express*, Vol. 12, pp. 4320-4325, 2004.

[46] K. Choi, H. Kim, and B. Lee, "Synthetic phase holograms for autostereoscopic image displays using a modified IFTA," *Opt. Express*, Vol. 12, pp. 2454-2462, 2004.

[47] K. Choi, H. Kim, and B. Lee, "Full-color autostereoscopic 3D display system using color-dispersion compensated synthetic phase holograms," *Opt. Express*, Vol. 12, pp. 5229-5236, 2004.

[48] S. W. Min, S. Jung, J. H. Park, and B. Lee, "Three-dimensional display system based on computer-generated integral photography," *Stereoscopic Displays and Virtual Reality Systems VIII, Proceedings of SPIE,* Vol. 4297, 2001. © 2001 SPIE · 0277-786X/01

[49] S. S. Athineos, N. P. Sgouros, P. G. Papageorgas, D. E. Maroulis, M. S. Sangriotis, N. G. Theofanous, "Physical modelling of a microlens array setup for use in computer generated IP," *Stereoscopic Displays and Virtual Reality Systems XII*, edited by A. J. Woods, M. T. Bolas, J. O. Merritt, I. E. McDowall, Proc. of SPIE-IS&T *Electronic Imaging,* SPIE Vol. 5664, 2005. SPIE and IS&T · 0277-786X/05

[50] A. Aggoun,"3D Visual information engineering (3D VIE)", School of Engineering and Design, Brunel University, UK, 2011.

[51] Y. Igarashi, H. Murata, M. Ueda, "3-D Display system using a computer generated integral photograph," *Japan J. Appl. Phys.,* Vol. 17, No. 9, 1978.

[52] A. Souchon, F. Tack, K. Christensen and E. Rabinovich, "Cylindrical lenses offer many focusing options", technology tutorial, www.cvimellesgriot.com. *Access date: Dec 2008.*

[53] J. E. Farrell, Brian L. Benson, and Carl R. Haynie, "Prediction flicker thresholds for video display terminals," *Hewlett Packard Pa10 Alto, CA, Proceedings of the SID,* Vol. 28/4, 1987

[54] D. David Wolff, "Overview of OpenGL 4.0 Shading Language Cookbook", Published by Packet Publishing, *USA, ISBN-10: 1849514763, 26 Jul 2011.*

[55] Anne Souchon, Francis Tack, Kevin Christensen and Emmanuel Rabinovich, "Cylindrical Lenses Offer Many Focusing Options", technology tutorial, www.cvimellesgriot.com. *Access date: Dec 2008.*

[56] Donald Hearn, M. Pauline Baker, "Computer Graphics with OpenGL", 3rd edition, *Published by Pearson Prentice Hall,* USA 2010

[57] http://www.opengl.org/, the formal OpenGL website. *Access date: 2012.*

[58] Cees van Berkel, John A Clarke, Philips Research laboratories, UK, "Characterisation and Optimisation of 3D-LCD Module design", *SPIE Digital Library,* Vol. 3012, 1997.

[59] Hiroyasu Ujike, Shin-ichi Uehara, “Human factors of autostereoscopic displays for standardization”, *Japanese Ergonomics National Committee, VIMS 2009*, 11th Jun 2009.

[60] http://images.apple.com/uk/quicktime/pdf/QuickTime7_User_Guide.pdf, *Access date: 2012.*

[61] R. F. Stevens, T. G. Harvey, “Lens arrays for a three-dimensional Imaging system”, Published 1 July 2002 Online at stacks.iop.org/JOptA/4/S17, *Access date: 2012.*

[62] T. Shirai, P. Johnson, “Three Dimensional (3-D) Displays in Japan”, *Tokyo*, 30 Apr 2006.

[63] H. Arimoto, B. Javidi, “Integral three-dimensional imaging with digital reconstruction”, *OPTICS LETTERS*, Vol. 26, No. 3, Feb 1, 2001.

[64] M. Martı´nez-Corral, B. Javidi, R. Martı´nez-Cuenca, G. Saavedra, “Integral imaging with improved depth of field by use of amplitude-modulated microlens arrays”, *2004 Optical Society of America, OCIS codes:* 110.6880, 110.4190, 350.5730, 2014.

[65] M. Martínez, B. Javidi, “Formation of real, orthoscopic integral images by smart pixel mapping”, *Optics Express*, Vol. 13, No. 23, 14 Nov 2005.

[66] J. Jang, B. Javidi, “Three-dimensional projection integral imaging using micro-convex-mirror arrays”, *Optics Express*, Vol. 12, No. 6, 22 Mar 2004.

[67] B. Javidi, “Integral three-dimensional imaging with digital reconstruction”, *U.S. Patent,* Pub. No. 0114077 A1, 22 Aug 2002.

[68] Stanley H. Kremen, “Method of forming a three-dimensional orthoscopic image from its pseudoscopic image”, *U.S.Patent*, Pub. No. 0122552 A1, 9 Jun 2005.

[69] A. Aggoun, “3D Holoscopic Imaging Technology for Real-Time Volume Processing and Display”, School of Engineering and Design, Brunel University, Uxbridge, UB8 3PH, (UK), 2010.

[70] Q. Wang, H. Deng, T. Jiao, D. Li, F. Wang, “Imitating micro-lens array for integral imaging”, School of Electronics and Information Engineering, Sichuan University, Chinese Optics Letters, Vol. 5, No. 5, May 10, 2010.

[71] Shree K. Nayar, “Computational cameras: Redefining the Image”, Columbia University, Computer IEEE, 0018-9162/06, 2006.

[72] Y. Frauel, B. Javidib, “Digital three-dimensional object reconstruction and correlation based on integral imaging”, IIMAS, University, D.F., Mexico, Electrical and Computer Engineering Dept., University. CT, USA, 2003.

[73] C. Fehn, “3D TV Broadcasting”, Fraunhofer Institute for Telecommunications, Berlin, Germany, 2001.

[74] J. Rosen and D. Abookasis, “Seeing through biological tissues using the fly eye principle”, Ben-Gurion University of the Negev Department of Electrical and Computer Engineering, Israel, OPTICS EXPRESS 3605 / Vol. 11, No. 26 / 29 Dec 2003.

[75] M. Ollis, T. Williamson, “The future of 3D video” Zaxel Systems Inc., Jun 2001.

[76] S. S. Athineos, N. P. Sgouros, P. G. Papageorgas, D. E. Maroulis, M. S. Sangriotis, N. G. Theofanous, “Photorealistic integral photography using a ray-traced model of capturing optics”, Vol. 15, No. 4, Oct–Dec 2006.

[77] D. E. Roberts, “History of lenticular and related autostereoscopic methods”, http://www.outeraspect.com/history_lenticular.php, Access date: 2012.

[78] A. BOGUSZ, “Holoscopy and holoscopic principles”, J. Optics (Paris), Vol. 20, 1989.

[79] P. Garbat, M. Kujawinsk, “Visuolization of 3D variable in time object based on data gathered by active measurement system”, Opto-Electron. Vol. 16, No. 1, 2008.

[80] D. E. Roberts, T. Smith, “The History of Integral Print Methods”, An excerpt from: Lens Array Print Techniques, 2003.

[81] S. M. Cirstea, S.Y. Kung, M. Mccormick, A. Aggoun, “3D-Object Space Reconstruction from Planar Recorded Data of 3D-Integral Images”, Journal of VLSI Signal Processing 35, 5–18, 2003.

[82] Ch. H. Wu, A. Aggoun, S. Y. Kung, “Depth measurement from integral images through viewpoint image extraction and a modified multibaseline disparity analysis algorithm”, Journal of Electronic Imaging, Vol. 14, No, 2, 023018, Apr–Jun 2005.

[83] J. Ren, A. Aggoun, M. McCormick, “Maximum viewing width integral image”, Journal of Electronic Imaging, Vol. 14, No. 2, 023019, Apr–Jun 2005.

[84] S. Manolache, A. Aggoun, M. McCormick, N. Davies, S. Y. Kung, “Analytical model of a three-dimensional integral image recording system that uses circularand hexagonal-based spherical surface microlenses”, J. Opt. Soc. Am. A, Vol. 18, No. 8, Aug 2001.

[85] B. Javidi, S. H. Hong, O. Matoba, “Multidimensional optical sensor and imaging system”, Applied Optics, Vol. 45, No. 13, 1 May 2006.

[86] F. Okano, J. Arai, “Optical viewer based on integral method for three-dimensional images”, Proc. of SPIE, Vol. 6392, 639201, 2006.

[87] B. Javidi, I. Moon, M. Daneshpanah, “3D imaging, visualization, and recognition of biological microorganisms”, Proc. of SPIE, Vol. 6392, 639202, 2006.

[88] B. Javidi, R. Martínez-Cuenca, G. Saavedra, and M. Martínez-Corral, “Orthoscopic, long-focal-depth integral imaging by hybrid method”, Proc. of SPIE, Vol. 6392, 639203, 2006.

[89] Y. Kim, H. Choi, J. Kim, Seong-Woo Cho, B. Lee, "Integral imaging with variable image planes using polymer-dispersed liquid crystal layers", Proc. of SPIE, Vol. 6392, 639204, 2006.

[90] R. Zaharia, A. Aggoun, M. McCormick, "Adaptive 3D-DCT compression algorithm for continuous parallax 3D integral imaging", Faculty of Computing Sciences and Engineering, De Montfort University, Queens Building, The Gateway, Leicester, LE1 9BH, UK, Signal Processing: Image Communication, Vol. 17, pp. 231–242, 2002.

[91] T. Georgeiv, K. C. Zheng, B. Curless, D. Salesin, Sh. Nayar3, Ch. Intwala, "Spatio-Angular Resolution Tradeoff in Integral Photography", Eurographics Symposium on Rendering, 2006.

[92] A. Aggoun, M. Mazri, "Wavelet-based compression algorithm for still omnidirectional 3d integral images", SIViP, DOI 10.1007/s11760-007-0044-1, Springer-Verlag London Limited 2007.

[93] B. T. Schowengerdt, E. J. Seibel, "True 3-D scanned voxel displays using single or multiple light sources", Extended version of a paper presented at the 2005 SID International Symposium held May 24–27, 2005, in Boston, Massachusetts, Journal of the SID, Vol. 14, No. 2, 2006.

[94] L. Onural, "Television in 3-D: What Are the Prospects?" Proceedings of the IEEE, Vol. 95, No. 6, Jun 2007.

[95] C. Smith, "On Vertex-Vertex Systems and Their Use in Geometric and Biological Modelling", Department of Computer Science, Calgary University, Alberta, Apr 2006.

[96] Fumio Okano Jun Arai M a koto Okui, "Three-dimensional Television using Integral Photography", 1-10-11, Kinuta, Setagaya-ku, Tokyo 1.57-8510, Japan, IEEE, 2001.

[97] H. Deng, Q. H. Wang, D. H. Li, "The Realization of Computer Generated Integral Imaging Based on Two Step Pickup Method", School of Electronics and Information Engineering, Sichuan University, Chengdu, China, IEEE, 2010.

[98] W. Li , Y. Li, Z. Hanjiang Yu, K. Wang, "Generic Calibration of an Integral Imaging Camera and its Applications on 2D Integral Image Reconstruction and 3D Scene Reconstruction", Proceedings of the 8th World Congress on Intelligent Control and Automation, Jinan, China, IEEE, Jul 6-9 2010.

[99] D. C. Hwang, K. J. Lee, S. C. Kim and E. S. Kim, "Extraction of location coordinates of 3-D objects from computationally reconstructed integral images basing on a blur metric", Optic Express, 3623, Vol. 16, No. 6, 17 Mar 2008.

[100] E. Niebler, "Boost.Accumulators", Distributed under the Boost Software License, Version 1.0, 2005-2006.

[101] N. Josuttis, "Boost.Array", Distributed under the Boost Software License, Version 1.0., 2001-2004.

[102] D. Gregor, “Boost.Function”, Use, modification and distribution is subject to the Boost Software License, Version 1.0, 2001-2004.

[103] Y. Igarashi, H. Murata, M. Ueda, "3-D Display system using a computer generated integral photograph," *Japan J. Appl. Phys.*, Vol. 17, No. 9, 1978.

[104] Roger Olsson and Mårten Sjöström, “A DEPTH DEPENDENT QUALITY METRIC FOR EVALUATION OF CODED INTEGRAL IMAGING BASED 3D-IMAGES”, This work is supported by the Swedish Graduate School of Telecommunications and by the EU Objective 1 - programme Södra Skogslän region.

[105] Roger Olsson and Mårten Sjöström, “A novel quality metric for evaluating depth distribution of artifacts in coded 3D images”, In Proceedings of Stereoscopic Display and Application XCIX, SPIE, Vol. 6803, San Jose (CA), USA, January, 2008.

[106] Anish Mittal, Anush K. Moorthy, Joydeep Ghosh and Alan C. Bovik, “ALGORITHMIC ASSESSMENT OF 3D QUALITY OF EXPERIENCE FOR IMAGES AND VIDEOS”, Dept. Of Electrical and Computer Engineering, The University of Texas at Austin, Austin, Texas - 78712.

[107] Andreas Abildgaard · Alaa KasidWitwit · Jørn Skaarud Karlsen · Eva Astrid Jacobsen · Bjørn Tennøe · Geir Ringstad. Paulina Due-Tønnessen, “ An autostereoscopic 3D display can improve visualization of 3D models from intracranial MR angiography”, Received: 25 March 2010 / Accepted: 15 June 2010 / Published online: 21 July 2010.

[108] Jie Shan, Chiung-Shiuan Fu, Bin Li, James Bethel, Jeffrey Kretsch, Edward Mikhail, “ AUTOSTEREOSCOPIC VISUALIZATION AND MEASUREMENT: PRINCIPLES AND EVALUATION”, Geomatics Engineering, School of Civil Engineering, Purdue University 550 Stadium Mall Drive, West Lafayette, IN 47907-2051, USA.

[109] Rafik Bensalma. Mohamed-Chaker Larabi, “Binocular Energy Estimation Based on Properties of the Human Visual System”, Received: 19 February 2012 / Accepted: 20 August 2012 / Published online: 13 September 2012.

[110] Weiming Li , Youfu Li, Zhanjiang Yu, and Keyi Wang, “Generic Calibration of an Integral Imaging Camera and its Applications on 2D Integral Image Reconstruction and 3D Scene Reconstruction”, 978-1-4244-6712-9/10/$26.00 ©2010 IEEE.

[111] Dong-Hak Shin and Hoon Yoo, “Image quality enhancement in 3D computational integral imaging by use of interpolation methods”, Dept. of Visual Contents, Dongseo University, San69-1, Jurye2-Dong, Sasang-Gu, Busan 617-716, Korea / Vol. 15, No. 19 / OPTICS EXPRESS 12039. 17 September 2007.

[112] Marcus Barkowsky, Romain Cousseau, Patrick Le Callet, « INFLUENCE OF DEPTH RENDERING ON THE QUALITY OF EXPERIENCE FOR AN AUTOSTEREOSCOPIC DISPLAY”, IRCCyN UMR 6597 CNRS, École Polytechnique de l’Université de Nantes,
rue Christian Pauc, La Chantrerie 44306 Nantes, France, 978-1-4244-4370-3/09/$25.00 ©2009 IEEE.

[113] Frank Pfenning, “Shading in OpenGL”, Carnegie Mellon University, February 14, 2002.

[114] Yu-Hsun Lin, Student Member, IEEE, and Ja-Ling Wu, Fellow, IEEE, “ Quality Assessment of Stereoscopic 3D Image Compression by Binocular Integration Behaviors”, IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 23, NO. 4, APRIL 2014

[115] Yun Sheng, Abdul H. Sadka, Senior Member, IEEE, and Ahmet M. Kondoz, Member, IEEE, “Automatic Single View-Based 3-D Face Synthesis for Unsupervised Multimedia Applications”, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 18, NO. 7, JULY 2008.

[116] David Fleet and Aaron Hertzmann, “Shading”, CSC418 / CSCD18 / CSC2504, 2005.

[117] Kim Daniel Skildheim, “Subjective and Objective Crosstalk Assessment Methodologies for Auto-stereoscopic Displays”, Norwegian University of Science and Technology Department of Electronics and Telecommunications, June 2012.

[118] Roger Olsson, “synthesis, coding and evaluation of 3D images based on integral imaging “, Department of information technology and media Mid Sweden University, Sundsvall, Sweden, 2008.

[119] Harit P Trivedi, and Sheelagh A Lloyd, “The Role of Disparity Gradient in Stereo Vision”, GECResearchLaboratories, Hirst Research Centre, Wembley, HA9 7PP, UK, Reprinted, with permission of Pion Ltd, from Perception, 1985, 14, 685-690.

[120] Jan J. Koenderink,”The Structure of Images”, Department of Medical and Physiological Physics, Physics Laboratory, State University Utrecht, The Netherlands Biol. Cybern. 50,363 370 (1984).

[121] Touradj Ebrahimi, “COST Action IC1003 – QUALINET European Network on Quality of Experience in Multimedia Systems and Services”, Multimedia Signal Processing Group Swiss Federal Institute of Technology.

[122] QUALINET European Network on Quality of Experience in Multimedia Systems and Services, “Qualinet White Paper on Definitions of Quality of Experience, Output version of the Dagstuhl seminar 12181”, Version 1.1 Dagstuhl, June3, 2012.

[123] Lutz Goldmanna, Francesca De Simone and Touradj Ebrahimi, Ecole Polytechnique Federale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland “A Comprehensive Database and Subjective Evaluation Methodology for Quality of Experience in Stereoscopic Video”, 2010.

[124] Marcus Barkowsky, Patrick Le Callet, Quan Huynh-Thu and Margaret Pinson, “VQEG 3DTV Group 3D Video Format Evaluation TEST PLAN In collaboration with DVB”, Draft Version 0.7, 31 October, 2014.

[125] Marcus Barkowsky, “VQEG 3DTV Group”, “Test Plan for Evaluation and Specification of Viewing Conditions and Environmental Setupfor 3D Video Quality Assessment”, 3DTV Viewing Con

itions Test Plan, Draft Version 1.0, 2012, 11/27/20141/19/2015.

[126] Taichi Kawano, “VQEG 3DTV Group”, “Test Plan for Evaluation of Video Quality Models for Use with Stereoscopic Three-Dimensional Television Content”, 3 DTV test plan, Draft Version 0.1 2012, 11/27/2014.

[127] Marcus Barkowsky, “VQEG 3DTV Group”, “Test Plan for establishing a Ground Truth for Quality of Experience in 3D for assessment methodologies in 3D Video Quality Assessment GroTruQoE3D1”, Draft Version 1.0, 2012, Ground Truth Quality of Experience 3D Test Plan, DRAFT version 1.0 27/01/2004. 11/27/2014.