

ANGLIA RUSKIN UNIVERSITY

FACULTY OF SCIENCE AND TECHNOLOGY

USE OF A HIGH RESOLUTION 3D OPTICAL
SCANNER FOR 3D MODEL CREATION, GAME
DESIGN AND FACIAL EXPRESSION RECOGNITION

GEORGIA CONSTANTINOU

A thesis is partial fulfilment of the requirements of Anglia
Ruskin University for the degree of Master of Philosophy

Submitted: April 2018

Acknowledgments

This dissertation was prepared in part fulfilment of the requirements of the degree of Master of Philosophy at Anglia Ruskin University. I would like to express huge gratitude to my first supervisor Dr George Wilson for his support, encouragement and motivation. I also gratefully acknowledge the funding I received from Anglia Ruskin University. I would also like to thank the Head of the Computing and Technology Department, Professor Marcian Cirstea for his support.

This work is dedicated to my family.

ANGLIA RUSKIN UNIVERSITY

ABSTRACT

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER OF PHILOSOPHY

USE OF A HIGH RESOLUTION 3D OPTICAL SCANNER FOR 3D MODEL
CREATION, GAME DESIGN AND FACIAL EXPRESSION RECOGNITION

GEORGIA CONSTANTINO

APRIL 2018

The process of three-dimensional (3D) scanning uses various techniques to capture the shape of an object in a computer file using a 3D scanner. This current research utilises a new camera-based 3D scanning technology (Mephisto Extreme 3D Optical scanner) that can very rapidly acquire high resolution 3D object models. The aims of this work include the configuration, assessment and evaluation of this 3D scanner to optimise scan quality, improve 3D object processing techniques that integrate 3D scanning, model construction and computer game development and evaluation of the use of the scanner for acquisition of facial features and its potential use in facial expression recognition. A procedure is presented detailing the configuration settings that will maximise 3D scan quality. The successful acquisition of numerous high quality 3D models from a variety of small inanimate and face target sources is reported. Appropriate graphics modelling software that can process 3D objects from acquisition and/or creation (Mephisto Extreme, 3DS Max) through to game import (Unity 3D) is presented and highlights the importance of facilitating portability of the 3D object models in the process chain. An OBJ file format reader/writer is developed where proof-of-principle is established that object model data output from the scanner can be easily and quickly extracted and potentially processed prior to input into a suitable game engine. This scanning-processing technique could potentially reduce the game design and development time from months/weeks to a few days. Other results include the successful scans of 3D facial expressions, and some possibilities for how this work could further progress research in 2D facial expression recognition are explored.

Keywords: 3D data, 3D scanner, OBJ loader, Facial expression recognition

Table of Contents

Acknowledgements	i
Abstract.....	ii
List of Figures	vi
List of Tables.....	viii
Copyright.....	ix
1. Introduction	1
1.1 Background.....	2
1.2 Types of 3D-Scanning.....	2
1.3 Aims and Objectives.....	3
1.4 Thesis structure.....	6
2 Review of 3D digitisation procedures.....	8
2.1 The representation of 3D objects.....	9
2.2 Polygonal mesh construction.....	12
2.3 3D Scanning technologies.....	17
3 3D scanner calibration, configuration and operation.....	21
3.1 Environmental and application considerations.....	22
3.2 Physical location and resolution issues.....	22
3.3 Starting a new Mephisto Extreme 3D Scanner Project.....	25
3.4 Adjusting camera settings.....	26
3.5 Calibration.....	27
3.5.1 Geometric calibration.....	28
3.5.2 Radiometric calibration.....	31
3.6 Calibration verification and the 3D-scanning of objects.....	33

3.7 3D scanning of objects.....	37
4 Object Modelling using 3D StudioMax.....	39
4.1 Object modelling versus object scanning.....	40
4.2 Architectural overview.....	40
4.3 The 3DS Max user interface.....	41
4.4 Creating/adding object primitives.....	43
4.5 Moving, rotating and scaling objects	44
4.6 Editing subsections of a 3D primitive.....	46
4.7 Copying objects.....	47
4.8 Saving files.....	47
4.9 Import/Export files on 3DS Max.....	48
4.10 Example of working with 3dsMax.....	49
5 3D objects in Unity3D.....	50
5.1 Unity3D for 3D object and game development.....	51
5.2 Unity3D projects and game development.....	51
5.3 The Unity User Interface (Unity UI).....	53
5.4 Navigating Unity.....	54
5.5 Game objects and asset creation.....	55
5.6 Importing and Exporting Unity assets.....	57
5.7 Examples of working with Unity 3D.....	60
6 Developing an OBJ file processing tool.....	61
6.1 3D Object data portability between applications.....	62
6.2 Features of Wavefront OBJ files.....	63
6.3 OBJ file structure.....	64
6.4 The polygonal geometry of a cube.....	66

6.5 Developing a C++ OBJ reader/writer.....	68
6.6 Summary of program development.....	73
7. 3D Scanning of a human face.....	74
7.1 Facial biometrics and emotional expression.....	75
7.2 Method of facial expression recognition.....	77
7.3 3D scanning of human facial expressions.....	79
7.4 Extending 2D facial expression recognition into the 3D domain.....	80
8. Conclusion.....	85
8.1 Summary of results.....	86
8.2 Evaluation of aims.....	86
8.3 Future work	89
8.4 Evolving technology.....	90
References.....	91
Appendix I: List of related research outputs.....	110
Appendix II: Source code.....	112
Appendix III: Electronic files.....	118

List of Figures

Figure 1.1: Description of the thesis flowchart.....	7
Figure 2.1: The surface-edge-vertex representation.....	14
Figure 2.2: Sample of 3D object with planar and cylindrical surfaces.....	15
Figure 3.1: Components of the Mephisto Extreme 3D Scanner.....	22
Figure 3.2: Calibration board.....	23
Figure 3.3: Set resolution.....	24
Figure 3.4: Starting a new project.....	25
Figure 3.5: Settings Tab.....	27
Figure 3.6: Set up for the geometric calibration.....	28
Figure 3.7: Geometric calibration results.....	29
Figure 3.8: How to perform radiometric calibration	31
Figure 3.9: Radiometric calibration results.....	33
Figure 3.10: The calibration board while scanning.....	35
Figure 3.11: The processing tab.....	35
Figure 3.12: Completed scan of calibration board.....	36
Figure 3.13: 3D model of the calibration board.....	36
Figure 3.14: Figurine with poor quality scan and high quality scan	37
Figure 4.1: The interface of 3ds Max.....	41
Figure 4.2: Viewport manipulation.....	42
Figure 4.3: Gizmo axis.....	45
Figure 4.4: Jet aircraft designed in 3DS Max by author.....	49
Figure 5.1: Basic Unity project.....	52

Figure 5.2: Adding a plane in the scene.....	54
Figure 5.3: The basic file structure of a Unity Project.....	57
Figure 5.4: The Project Window shows assets that have been imported into your project.....	58
Figure 5.5: The relationship between the Assets Folder in a Unity Project on a computer and the Project Window within Unity.....	59
Figure 5.6: Screen capture of Unity 3D game scenario in development.....	60
Figure 6.1: Left: ASE file format for a cube; Right: OBJ file format for the same cube.....	63
Figure 6.2: Positioning of cube on pedestal in preparation for 3D scanning.....	66
Figure 6.3: Screen capture of 3D data output from file ‘pinkbox.obj’.....	72
Figure 7.1: Illustrative image of face detection.....	75
Figure 7.2: Six basic emotions and their typical associated expressions	76
Figure 7.3: 3D models of facial expressions acquired by the Mephisto 3D scanner.....	80

List of Tables

Table 2.1 List of 3D modelling software.....	10
Table 3.1 Comparison of two result sets for low quality versus high/medium quality geometrical calibration.....	30
Table 3.2 Inventory of objects scanned by Mephisto 3D scanner.....	38
Table 4.1 Import and Export file formats for 3dsMax.....	48
Table 5.1 Mac Unity Hotkeys.....	56
Table 6.2 Comparative summary 3D data for various object files.....	73
Table 7.3 FACS action units (AU).....	82
Table 7.4 Examples of combination of FACS action units.....	82

Copyright

Copyright Attention is drawn to the fact that copyright of this thesis rests with

- (i) Anglia Ruskin University for one year and thereafter with
- (ii) Georgia Constantinou

This copy of the thesis has been supplied on condition that anyone who consults it is bound by copyright.

This work may:

- (i) be made available for consultation within Anglia Ruskin University Library,
or
- (ii) be lent to other libraries for the purpose of consultation or may be
photocopied for such purposes;
- (iii) be made available in Anglia Ruskin University's repository and made
available on open access worldwide for non-commercial educational
purposes, for an indefinite period.

Chapter 1:

Introduction

1.1 Background

The process of three-dimensional (3D) scanning uses a technique to capture the shape (3D-geometry, or mesh data) of an object in a computer file using a 3D scanner. That data can be viewed graphically, processed/edited, or 3D printed. In recent years this technology has advanced at a rapid pace in terms of modes and speed of acquisition, hardware portability and ease of use (Weise et al, 2007). Traditionally, such technology has mainly been used in surveying, medicine and engineering (Frohlich et al, 2004; Thali et al, 2009; Kus et al, 2009) but the complimentary reduction in cost associated with these new technological advances has opened up new opportunities for 3D-scanning applications in forensic science, archaeology (including small artefact archiving) and for animation including film and computer games (Kuzminsky, 2012; Lerma, 2010; Tong et al, 2012). In 2012 Anglia Ruskin University acquired a Mephisto Extreme (Ex-PRO) Optical 3D-scanner¹ housed in the Department of Computing and Technology at ARU Cambridge. At the time of procurement this device was capable of unprecedented scan resolution for a fraction of the scan time duration conventionally associated with 3D-scanning (4D Dynamics, 2012). This current research explores some new potential applications of small artefact scanning (typically 0.1-0.5m³) using this equipment especially with respect to the gaming industry and acquisition of 3D models of the face.

1.2 Types of 3D-Scanning

There are four basic types of 3D scanning, listed below;

- **Laser scanning** – there are two variations of this type of scanning. Firstly, **Time of Flight** 3D-scanning (also referred to as a laser pulse) involves projecting a laser

¹ Purchased from Inition (<https://www.inition.co.uk/mephisto-3d-scanning-engine/>)

beam onto a surface which is reflected back to a sensor; the time-of-flight between emission and reception provides geometrical information about the surface (Pallone et al, 2016). Secondly, **Laser Triangulation** projects a laser beam onto a surface and measures the deformation of the laser rays on that surface (Kjaer et al, 2015).

- **Photogrammetry** – 2D images are taken at different viewpoints of the same scene or object and are used to compute a 3D data set using principles of stereoscopic vision. Remotely-sensed satellite images for example are now routinely used to construct landscape topography (Luhmann et al, 2014; Bemis et al, 2014).
- **Contact-based scanning** – a probe is used to touch a number of points on a surface and it records the deformation (Celano et al, 2015).
- **Structured light scanning** – a pattern of light is projected on to a surface and its deformation is used to construct the 3D shape of the surface (Garrido-Jurado et al, 2016).

The Mephisto Extreme used in the present study is an optical (camera-based) scanner, combining the use of structured lighting and photogrammetry. One camera is used to acquire the geometrical information from a repeatedly rotated/repositioned target object whilst a second camera can optionally be used to simultaneously acquire textural information (that is, information about the colour of the surface).

1.3 Aims and Objectives

The fundamental aim of this work is to evaluate the capabilities of the Mephisto Extreme in enhancing the development process for a variety of applications. The chosen

applications are based on the current research focus of the environment the researcher is based, both in terms of expertise and hardware technology. The identified aims of this work, each utilising the Mephisto Extreme will now be explained.

Aim 1: Configure, assess and evaluate a new 3D scanning technology (Mephisto Extreme 3D Optical scanner) as an improved technique for acquiring rapid high resolution 3D object models.

The Mephisto Extreme specification should allow the capture of objects to a very high level of mesh detail in time frames that are orders of magnitude faster than either the traditional rendering software design tools and faster than other conventional 3D scanning techniques. A 3D model can be produced in less than 5 minutes compared to the usual drawing or sculpting procedure which would take days or even weeks for the same model to be produced. The contribution to knowledge for this aim will be to evaluate a new (at the time of writing) 3D scanning technology from an operational point of view.

Aim 2: Improve 3D object processing techniques that integrate 3D scanning, model construction and game development.

A major bottleneck in game development is the time involved to design 3D models for computer games (Foster et al, 2016). A game designer would typically construct a virtual game artefact using a modelling software package such as 3D-StudioMax, a process that can take weeks. The 3D model is then imported into a game development environment for further work. An alternative strategy for some game artefacts is to 3D-scan a physical object for game import. The contribution to knowledge for this aim will be to evaluate

the limitations and possibilities for this optical scanning technology for the game development process.

Aim 3: Evaluate the use of the optical scanner in the acquisition of facial features and its potential use in facial expression recognition.

The 3D scanning of a human face provides biometric data that has many applications especially for medical and for security (identification) purposes (Smeets et al, 2010). Given the high resolution capability of the Mephisto Extreme this facility will be used to 3D scan some example target faces where various facial expressions are presented to the scanner. The contribution to knowledge will be to assess the potential use of such 3D data for facial expression recognition in conjunction with existing 2D data sets.

The three aims will be achieved through the attainment of a set of specific objectives, listed as follows:

Objective 1: Evaluate the various 3D scanning techniques, assessing the pros and cons of 3D optical scanning (that being the equipment used in the present study).

Objective 2: Calibration of the Mephisto Extreme control parameters and settings to enable rapid, high quality 3D scans of small objects (0.1-0.5m³).

Objective 3: Successfully scan a number of different shapes of various surface geometrical complexities, colour, reflectivity and transparency using the Mephisto Extreme 3D scanner.

Objective 4: Demonstrate use of a graphics modelling software package in creating 3D mesh models and how the techniques can be equally applied to 3D objects created by design and 3D objects acquired by 3D scanning.

Objective 5: Demonstrate use of a game development software package in both creating 3D mesh models and importing such models either acquired by other software packages or by 3D scanning.

Objective 6: Develop a file format specific reader/writer for displaying, loading and saving 3D mesh data, where access to mesh data is demonstrated as proof-of-principle that such an application could be used to process geometrical data.

Objective 7: Consider the potential applications of high resolution 3D scanning of the human face for facial expression recognition.

These objectives approximate (but do not exactly coincide) with the chapter structure of this thesis.

1.4 Thesis structure

Chapter 1 (this chapter) describes the aims and objectives of this research, including a rationale for those aims. Chapter 2 presents a detailed review of 3D digitisation procedures, compares some common 3D modelling software applications and introduces the principal hardware tool used in this current work – the Mephisto Extreme 3D Optical Scanner. Chapter 3 provides a detailed description of the Mephisto Extreme 3D-scanner calibration, configuration and operation and how 3D object meshes are acquired and saved to file. Chapter 4 focusses on 3DS Max, a computer graphics modelling environment widely used for commercial and academic purposes to create and edit 3D objects, whether designed or imported. Chapter 5 follows on from the previous chapter

in describing the manipulation of 3D objects in Unity3D, a common computer game building software development environment. Chapter 6 presents the software development aspects of this research, specifically the development of a program for use between object acquisition/creation and import into a game environment through manipulation of a file reader/writer for the OBJ file format. Chapter 7 considers one specific other application of 3D scanning in its potential impact for facial expression recognition. Chapter 8, the conclusion, presents the main results of this work, assesses the extent to which the aims have been achieved, and considers some possibilities for future work. This is displayed on the flowchart below (Figure 1.1).

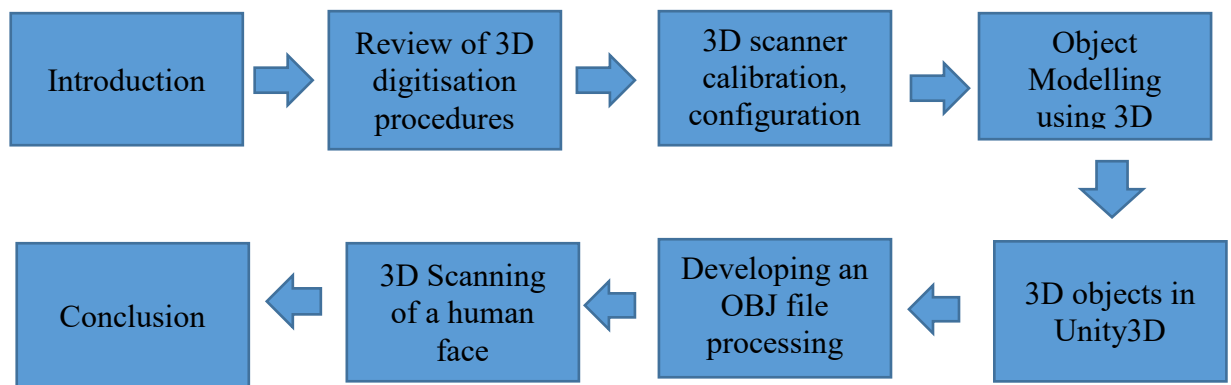


Figure 1.1: Description of the thesis flowchart

To conclude, this chapter has described the aims and objectives of this research, including a rationale for those aims as well as the thesis structure.

Chapter 2:

Review of 3D digitisation procedures

2.1 The representation of 3D objects

In computer graphics, 3D objects of the real and the imaginary world can be expressed in numerical form as 3D models (Zhao et al, 2012). The computational representation of the model surfaces is a widely studied field (Roncat et al., 2011) and may comprise very large data sets which describe geometrical and appearance attributes (Finney, 2013). The 3D models can be produced using interactive software development packages (e.g. 3D Studio Max, Maya) but introduce a major bottleneck in the design and development process and often produce 3D models in too much detail for practical use.

The surface data of an object is often initially reported in the form of irregularly spaced X, Y, and Z geometric coordinates stored in an appropriate data structure. This collection of points is referred to as a point cloud. Such point clouds can be acquired from hardware sensors such as stereo cameras, 3D scanners, or generated from a computer program synthetically (Radu and Cousins, 2011). Point cloud data can be represented by a collection of vertices, edges, and faces, which are used to generate complex polygonal meshes to represent the models in virtual space. The polygonal mesh defines the shape of an object. Every face of a mesh is an addition of triangles and so a face must consist of at least three vertices respectively (Tobler and Maierhofer, 2006). In some surface interpolation techniques a three dimension mesh could be compared to a Triangular Irregular Network (TIN) locally.

There is a range of commercial and open-source 3D modelling platforms some of which offer 3D rendering support. Table 2.1 lists some of these at the time of writing.

Table 2.1: List of some common 3D modelling software applications*

Title	License	3D rendering support
3D-Coat	Commercial software	Yes
3D Slash	Freemium	Yes
3DVIA Shape	Commercial software	No
AC3D	Commercial software	No
Anim8or	Freeware	Yes
Animation:Master	Commercial software	Yes
Art of Illusion	GNU GPL	Yes
AutoCAD	Commercial software	Yes
AutoQ3D Community	GNU GPLv2 +	Yes
AutoQ3D	Commercial software	Yes
Autodesk 123D	Freeware	No
Autodesk 3ds Max	Commercial software	Yes
Autodesk Inventor	Commercial software	Yes
Autodesk Maya	Commercial software	Yes
Autodesk Revit	Commercial software	Yes
Autodesk Softimage	Commercial software	Yes
Blender	GNU GPLv2+	Yes
BRL-CAD	GNU LGPL and BSD	Yes
Bryce	Commercial software	Yes
CATIA	Commercial software	Yes
Carrara	Commercial software	Yes
Cheetah3D	Commercial software	Yes
Cinema 4D	Commercial software	Yes
CityEngine	Commercial software	No
Clara.io	Freemium	Yes
DAZ Studio	Freemium	Yes
Electric Image Animation System	Commercial software	Yes
Flux	Freeware	No
Form-Z	Commercial software	Yes
fragMOTION	Commercial software	No
FreeCAD	GNU LGPL	Yes
Hexagon	Commercial software	No
Houdini	Commercial software	Yes
Leapfrog3D	Commercial software	Yes
LightWave 3D	Trialware	Yes
MASSIVE	Commercial software	No

Metasequoia	Commercial software	Yes
MikuMikuDance	Freeware	Yes
MilkShape 3D	Commercial software	No
Modo	Commercial software	Yes
Moi3D	Commercial software	Yes
Open CASCADE	GNU LGPL	Yes
OpenSCAD	GNU GPL	Yes
Poser	Commercial software	Yes
Pro/ENGINEER	Commercial software	Yes
Quake Army Knife	GNU GPL	Yes
RaySupreme	Commercial software	Yes
Realsoft 3D	Commercial software	Yes
Remo 3D	Commercial software	Yes
Rhinoceros 3D	Commercial software	Yes
SOCET SET	Commercial software	No
Sculptris	Freeware	No
Seamless3d	MIT	No
Shade 3D	Commercial software	Yes
Silo	Commercial software	No
Sketchup	Freemium	Yes
Solid Edge	Commercial software	Yes
solidThinking	Commercial software	Yes
SolidWorks	Commercial software	Yes
SpaceClaim	Commercial software	Yes
Strata 3D	Commercial software	Yes
Swift 3D	Commercial software	No
TopMod	GNU GPL	No
TrueSpace	Freeware	Yes
Unigraphics	Commercial software	No
Wings 3D	BSD	Yes
Vectorworks	Commercial software	Yes
ZBrush	Commercial software	Yes
Zmodeler	Freeware	Yes

*

* URL: https://en.wikipedia.org/wiki/List_of_3D_modeling_software) accessed 12th March 2018

In addition to the listing in Table 2.1, some commercial examples offer innovative solutions for three dimension mesh generation, visualisation, interpretation and analysis, such as Leica Cyclone, 3DReshaper or Geomagic Wrap (Pucci and Marambio, 2009; Roncat et al., 2011). Comparable open-source solutions include MeshLab and CloudCompare. In particular MeshLab software[◇] is a free, open-source software for mesh processing and editing and which generates a triangular 3D-mesh (Cignoni et al., 2008). This software works with the most common 3D file formats, such as OBJ, 3DS, PLY, STL, COLLADA, XYZ, ASC, X3D, PTX, PTS, XYZ, ASC, X3D and VRML. MeshLab is a principle software application package used in the current research along with 3Ds Max, and has a range of algorithms which can be used to reconstruct surfaces from point clouds. Common to many computer graphic tools these applications can generate surface meshes composed of 10 - 100 M faces. The Mephisto Extreme 3D scanner used in the present work typically produces very complex meshes with a high level of detail. Mandatory simplification procedures are often necessary to reduce the mesh size to allow easier management and use in other applications. A few techniques produce simplified meshes which are visually undesirable but offer efficient processing time (Alvarez et al, 2007). Two very common simplification operations are the edge collapse (Zhang, 2013, Ma et al, 2012) and vertex collapse (Attali et al, 2013).

2.2 Polygonal mesh construction

Most graphics hardware in computer graphics and computer vision support 3D mesh construction as it is the most common kind of structure used for rendering and display. Representations other than a polygonal mesh include B-spline surfaces, quadric surfaces and subdivision surfaces for smoother or simpler surfaces. A graphics representation can

[◇] <http://meshlab.sourceforge.net>

also contain colour as well as texture information which can “texture-mapped” onto the object which will be rendered by the graphics hardware, sometimes through use of images. This is important in computer vision, where there must be some kind of correspondence between the 3D object and the characteristics in the texture mapped image. This is particularly important for 3D object recognition, utilising grey-scale images, colour images, and range images. In addition to this, it is very common to have either grey-scale or colour images registered to range data, therefore providing recognition algorithms with richer sets of characteristics.

A wire-frame model is a three dimensional (3D) object model which consists of only the edges and vertices of the object. The surfaces of the wire-frame representation of the object are assumed to be planar and that the object has only straight edges. The surface-edge-vertex representation is a useful generalisation of the wire-frame model that has been used in computer vision. This representation is a data structure which contains the vertices, the surfaces, the edge segments of the object, as well as the topological relationships that specify the surfaces on both sides of an edge and the vertices on both ends of an edge segment. The surfaces of a polygonal object are planar and the edge segments are straight line segments. However, the model generalises to include curved edge segments and/or curved surfaces. Figure 2.1 presents a sample of a surface-edge-vertex data structure which was used for illustrating a database of object models in a 3D object recognition system. The data structure is hierarchical. It begins with the world at the top level and continuing down to the surfaces and arcs at the lowest level. The boxes with fields labelled [name, type, <entity>, transf] describe the elements of a set of class <entity>. Each element of the set has a name, a type, a pointer to an <entity>, and a 3D transformation that is applied to the <entity> to obtain a potentially rotated and translated instance.

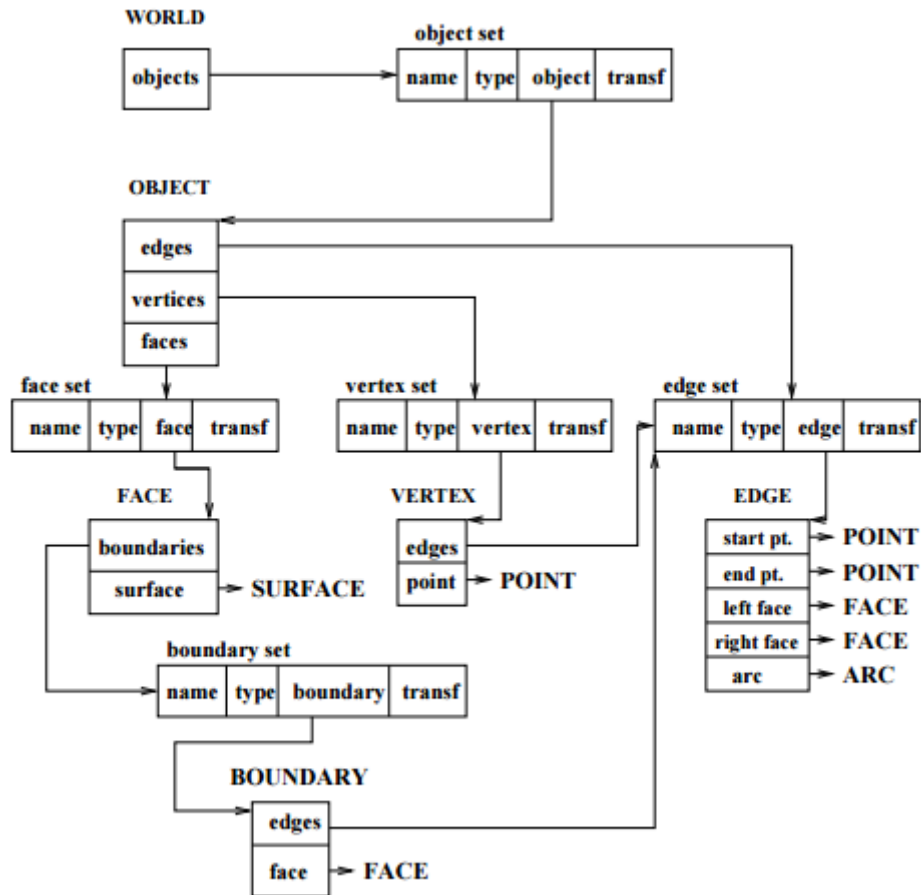


Figure 2.1: The surface-edge-vertex representation (Shapiro and Stockman, 2000).

For example, the world has a set called objects. In that set are named instances of various 3D object models. Any given object model can be defined in its own coordinate system and the transformation allows each instance to be placed in the world independently. The object models each have three sets: their edges, vertices and faces. (Shapiro and Stockman, 2000). A vertex has a 3D point it is associated with and it also has a set of edges that meet at that point. An edge on the other hand, has a start point, an end point, a face to its left, a face to its right and an arc that denotes its form, if it is not a straight line. A face has a surface that describes its shape and a set of boundaries which are its outer boundaries and hole boundaries. A boundary has an associated face and a set of edges. The lowest level entities (arcs, surfaces, and points) are not defined. The representations for surfaces and arcs are

depending on the application as well as the accuracy and smoothness that is required. The representations might be presented with equations or broken down into surface patches and arc segments. The points are merely vectors of x, y, z coordinates.

Figure 2.2 demonstrates a sample of 3D object with planar and cylindrical surfaces. In order to simplify the demonstration, a small number of visible surfaces and edges will be analysed. F1, F2, F3, F4, and F5 are the visible surfaces that will be discussed. As shown in Figure 2.2, F1, F3, F4, and F5 are planar surfaces. F2 is a cylindrical surface. F1 is has a single boundary composed of a single edge and it can be demonstrated by a circular arc. F2 is surrounded by two boundaries. F3 is enclosed by an outer boundary which is suppressed of four straight edges and a hole boundary suppressed of a single circular arc. F4, and F5 are each enveloped by a single boundary possessed of four straight edges. Edge E1 detaches faces F3 and F5. If vertex V1 is a start point and V2 is the end point, then F3 is its left face and F5 is the right face. Vertex V2 has three connected edges E1, E2, and E3 (Shapiro and Stockman, 2000).

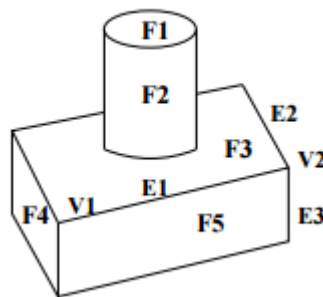


Figure 2.2: Sample of 3D object with planar and cylindrical surfaces (Shapiro and Stockman, 2000).

As is implied by Figure 2.2, complex shapes can be abstracted by assembling volumetric primitives; some of the more common primitive models are as follows:

- Generalised-Cylinder Models - A generalised cylinder is a volumetric primitive defined by a space curve axis as well as a cross section function at each point of the axis. This cross section is swept along the axis creating a solid of revolution. A common circular cylinder is for example, a generalised cylinder whose axis is a straight line segment and its cross section is a circle of constant radius. A cone can be considered to be a generalised cylinder whose axis is a straight line segment and whose cross section is a circle whose radius starts out from zero at one end point of the axis and grows to its maximum at the other end point. A rectangular solid can be a generalised cylinder whose axis is a straight line segment and whose cross section is a constant rectangle. A torus is a generalised cylinder whose axis is a circle and whose cross section is a constant circle.
- Octrees - An octree is considered to be a hierarchical 8-ary tree structure. Each node in the tree is associated to a cubic region of the universe. If the cube is completely enclosed by the three dimensional object then the label of a node is full (1) and it is empty (0) if the cube does not contain part of the object, or partial, if the cube intersects the object partly. A node with full or empty label has no children. A node with partial label has eight children showing the partition of the cube into octants. A 3D object can be represented by a $2n \times 2n \times 2n$ 3D array for some integer n . The elements of the array have a value of 1 (full) or 0 (empty) and they are called voxels. These elements are demonstrating the existence or the absence of the object. The octree encoding of the object is equivalent to the 3D array representation, but generally requires less space.

- Superquadrics - these are models originally refined for computer graphics and recommended for use in computer vision (Barr, 1981). Superquadrics can be considered as pieces of clay that can be reformed and glued together into object models. They can also form a parameterised family of shapes. A superquadric surface is defined by a vector S whose x , y , and z components are specified as functions of the angles η and ω via the equation:

$$S(\eta, \omega) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_1 \cos^{\epsilon_1}(\eta) \cos^{\epsilon_2}(\omega) \\ a_2 \cos^{\epsilon_1}(\eta) \sin^{\epsilon_2}(\omega) \\ a_3 \sin^{\epsilon_1}(\eta) \end{bmatrix}$$

for $-\frac{\pi}{2} \leq \eta \leq \frac{\pi}{2}$ and $-\pi \leq \omega < \pi$.

The parameters a_1 , a_2 , and a_3 specify the size of the superquadric in the x , y and z directions, respectively. The parameters ϵ_1 and ϵ_2 express the squareness in the latitude planes and longitude planes.

2.3 3D Scanning technologies

3D scanning technologies are an important aid in product development, and creating good digital representations of an artefact is often a crucial process during the manufacturing method. 3D scanners are used to digitise objects that are from microscopic to macroscopic in size. Devices can be hand-held or automatic, with varying acquisition speeds and data point resolution (Schodek et al, 2005).

There are two main methods for obtaining the coordinates of an object's geometrical shape.

1. Mechanical method - the object is fixed on a table and the coordinates of the points are picked by a human inspector by means of touch-probes which transfer the points to a computer. Measurement may take hours or even days depending

on the details of the object and accuracy of the measurement required. Accuracy levels up to 1 μm can be achieved by using this method. This level of sensitivity depends on the experience of the inspector and type of the equipment used.

2. Non-contacting scanning methods – these include acoustic, magnetic and optical methods.

For the scanning of small objects optical technology is the generally preferred method because it gives a greater flexibility in the digitisation of surfaces and provides a higher capture resolution and accuracy at greater speeds when compared to mechanical technology (Blais, 2003; Li, 2002; Milroy, 1996; Sokovic, 2006; Tognola, 2003). Optical scanners are also more portable compared to contact systems and their sensitivity levels are partially independent of the inspector. However, relevant to the current research one advantage of contacting devices over optical scanners is that they do not depend on the colour and reflective characteristics of the surfaces to be scanned.

Optical scanning systems for the 3D measurement and virtual reconstruction of object surfaces are based on techniques such as laser scanning, fringe projection, and photogrammetry. Fringe projection scanning systems generally work with white structured light, where the light pattern is projected on the object's surface while one or two cameras record the reflected light. Laser scanning systems on the other hand obtain data by sending laser light onto the object and processing the data obtained from the returning light (Bernard, 1999, Varady et al, 1997, Peipe et al, 2005). The accuracy of laser systems varies typically from 1 - 20 μm , whereas fringe projection systems have the capability of 10 - 60 μm , and this accuracy is continually improving (Seokbae et al, 2002).

The current research utilises the Mephisto Ex-PRO Optical scanner (also referred to as the Mephisto Extreme). This is a variation on the fringe projection scanning technology and can achieve fast, accurate and high quality 3D scanning results with a minimum of processing time and user interaction, competitive with the highest range of alternative scanners on the market at the time of writing. It is suitable for a variety of scanning jobs from the very small to the scanning of large animated as well as static objects.

The Mephisto Extreme hardware consists of:

- a deep scan, turntable mode, HDTV (1920 x1080) resolution machine vision camera with large, high quality Kodak CCD sensor
- a high quality Nikon mount lens with aspheric optics, high contrast ratio high resolution projector(2000-3000 ANSI lumens with 2000:1 contrast ratio; wide screen format and 1280x768 resolution). The projector is a Digital Light Processing (DLP) - type for use with binary and fringe pattern projection. DLP projectors provide an excellent base for a structured light 3D scanning system.

All the scanner components are synchronised and linked together by a cable assembly and Mephisto software. In principle the system is portable to allow off-site scanning.

The Mephisto 3D scanning engine is based on 3 core components – calibration, processing and the input/output interface. The scanner geometric system calibration is performed using a flat calibration board with a checker pattern. 5 to 6 shots snapshots of the calibration board at different orientation angles are taken and used to compute system intrinsic and extrinsic parameters. The Mephisto software is project-based and a wizard

helps the user to define the properties of scanning system such as camera(s), resolution/texture, and projection screen. Once a project is created and the system is calibrated, the project can be repeatedly saved and/or reloaded with the project saving scanned data, images and other information in an organized manner within a few seconds or minutes.

This chapter presented a detailed review of 3D digitisation procedures, compares some common 3D modelling software applications and introduces the principal hardware tool used in this current work – the Mephisto Extreme 3D Optical Scanner.

Chapter 3:

3D scanner calibration, configuration and operation

3.1 Environmental and application considerations

This chapter will describe the procedure of setting up the 3D Mephisto scanner for optimum scan quality in terms of both environmental (physical) set up and application (software) configuration. Some calibration results will be presented and discussed.

3.2 Physical location and resolution issues

For calibration purposes the 3D scanner requires a dedicated dark room in which the equipment can be arranged in a stable position. Whilst in theory the whole unit is portable (i.e. for use in temporary deployment outdoors), in practice for this study a stable environment is important to facilitate application usage, otherwise time-consuming environmental and recalibration activities hinder research development time. The 3D scanner has four important physical components as show in Figure 3.1:

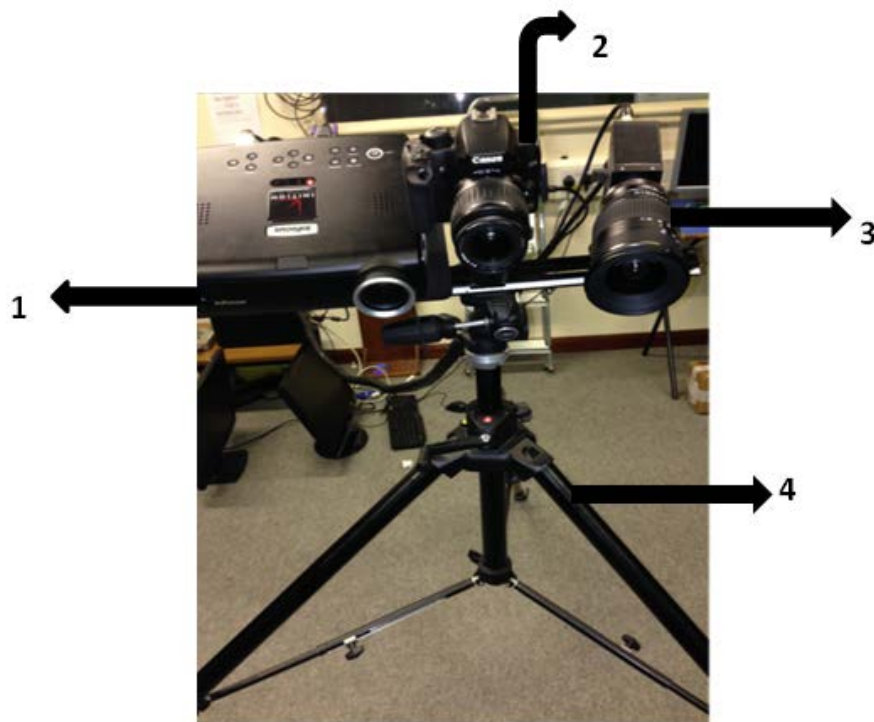


Figure 3.1: Components of the Mephisto Extreme 3D Scanner

Where:

1. Projector Digital Light Processing (DLP) unit. This acts as a light source with a resolution of 1280x800 at 60Hz and a brightness of 2000-3000ANSI Lumens. The unit enables much faster acquisition than a laser scanner.
2. A Texture Digital Canon camera 400D.
3. A digital camera Allied Vision Technology (AVT) PIKE.
4. Flexible tripod mount.

A calibration board and pattern is also supplied which is fundamental to the calibration process (Figure 3.2):

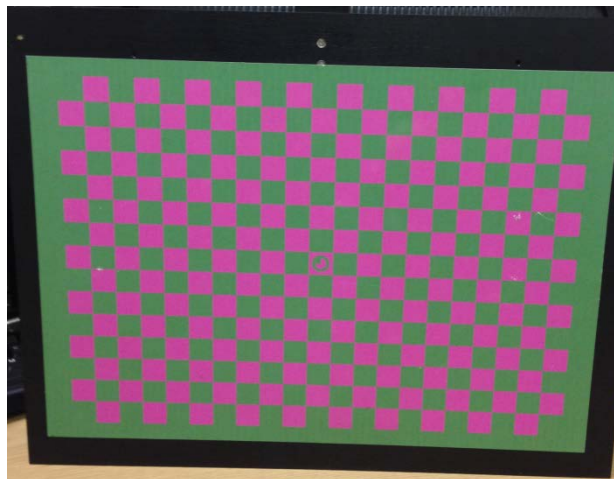


Figure 3.2: Calibration board

A detailed description of the hardware set-up parameters is described in a Quick Start guide for the 3D scanner*. These parameters are extrinsic (e.g. camera and projector position) and intrinsic (e.g. projector and camera lens properties). The inflexible calibration board is prepared by printing out a supplied checker pattern on non-reflective matte paper which is attached to the board, and positioned perpendicular to light that will be emitted from the projector. An important preliminary step is to ensure the scanner

* Mephisto Extreme Quick Start Guide: 4D Dynamics (www.4ddynamics.com)

resolution matches the resolution of the PC screen monitor (otherwise the scanner will not function correctly). For 3D scanner projects specific to this work (Windows XP) the following steps were taken:

1. Right-click the Desktop
2. Click on NVIDIA Control Panel
3. Select “Change resolution” (Figure 3.3)
4. Choose the projector
5. Check resolution ensuring it is “Native resolution”
6. Set resolution to 1280 x 800 and refresh rate is 60Hz

Note use of different peripheral hardware (e.g. different graphics card, monitor) might require different settings to be used.

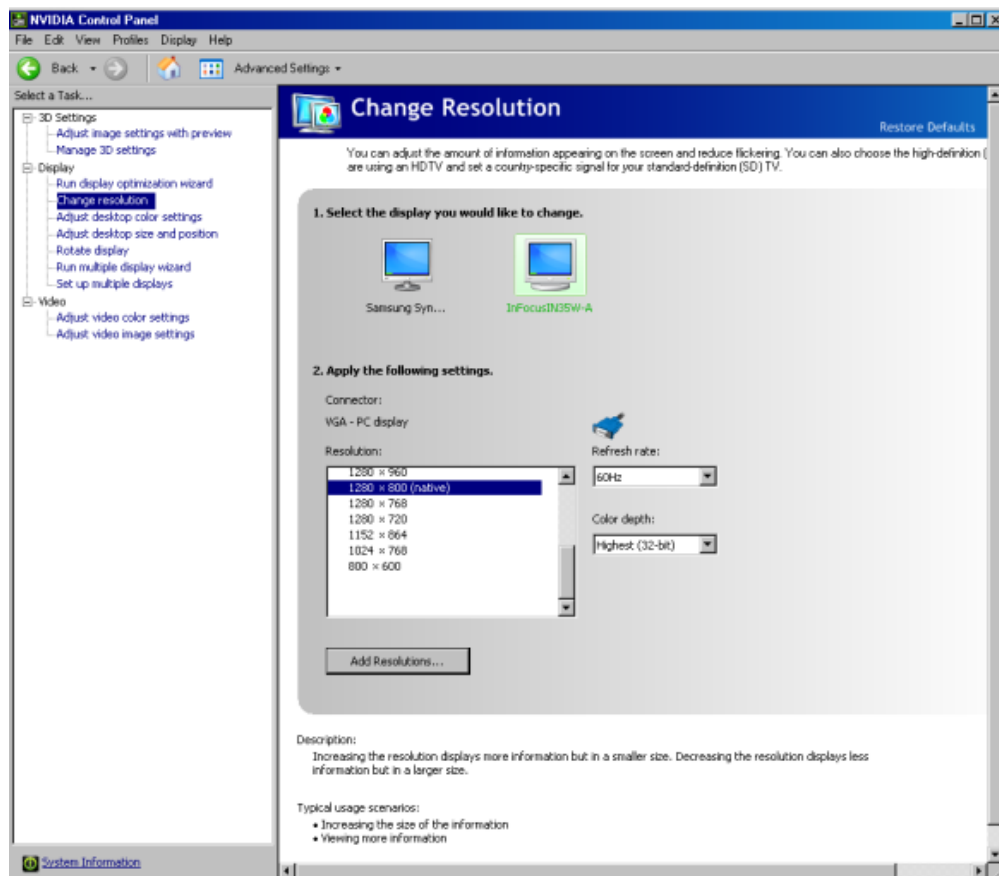


Figure 3.3: Set resolution

3.3 Starting a new Mephisto Extreme 3D Scanner Project

Once projector and monitor resolution have been synchronised the projector and both of the cameras can be switched on. The Mephisto Extreme software icon is then clicked to launch the application (Mephisto Extreme software version 1.6.1213). A 'New project wizard' window opens (Figure 3.4) and information is entered as shown.

New project wizard - Step 1 of 1

Project information
Camera/Projector system selection

Project name: myProject15-08

Base directory: C:/My scanner

Camera selection: 3D reconstruction

☒ Firewire camera: AVT - Pike F2108

☐ Canon camera: <select canon device>

☐ Trigger on serial port: <select serial port>

☐ Prerecorded frames: ...

☐ Camera selection: high-quality texture generation

☐ Firewire camera: <select firewire device>

☐ Canon camera: <select canon device>

☐ Trigger on serial port: <select serial port>

☐ Prerecorded frames: ...

Projector settings

Projector is hooked onto screen: Screen 2 @ 1280x800

☐ Preload radiometric data: C:/Mephisto Extreme/myProjecttest/BRY015Csuccessful.mpl

Calibration Checkers Patterns

Board: # squares (WxH): 21 x 15 of size: 18.00 mm

Projector: # squares (WxH): 21 x 15 dimensions: 1024 x 768

Cancel < Back Next > Finish

Figure 3.4: Starting a new project

Project name: Enter an appropriate name for the project, e.g. inclusive of month/year.

Base directory: Save a new Project the C:/ drive rather than the default software directory as this keeps data separate from the application software and enables easier file management at a later time if necessary.

Camera selection: For projects for this work the Firewire camera: AVT- Pike F210B is chosen unless high quality texture generation is required in which case the Canon Camera is chosen.

Project settings: Ensure the 'Projector is hooked onto the screen:' entry specifies Screen 2 @ 1280x800 (the same as the native resolution specified for the computer display). The 'Preload any radiometric data' is not applicable in this case so the checkbox can be left blank.

Calibration Checkers Patterns: Here the user can modify the number of the squares on the calibration pattern. For this work the default options are used, i.e. 21x15 squares for both the board and projector with an 18mm size.

Clicking 'Finish' closes the wizard. The cameras will be activated and the main Mephisto interface will open in a new window.

3.4 Adjusting camera settings

The main Mephisto interface has three tabs; 'Settings' for modification of camera properties, 'Calibration' for geometric and radiometric calibration and 'Processing' for the scanning of target objects. The 'Settings' tab allows configuration of both the projector and the Firewire camera. The default settings are usually in factory default mode but the user has the flexibility to change them.

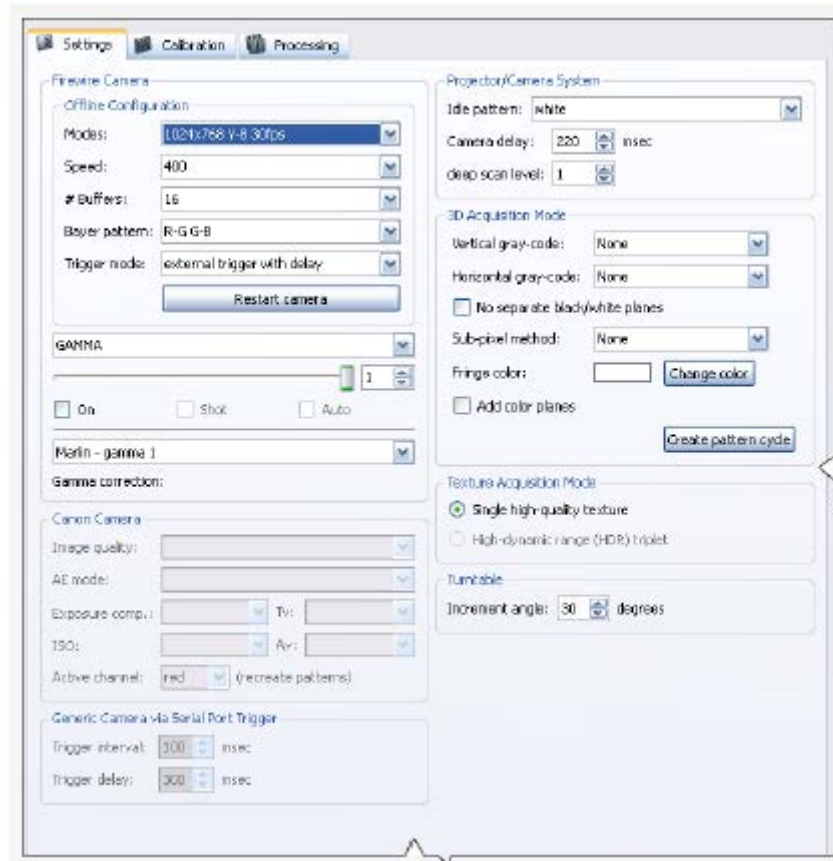


Figure 3.5: Settings Tab

For projects for this work default parameters were used except for the Firewire camera (Online configuration Modes: HDTV resolution; Shutter speed: 823; and Parameters: GAIN).

3.5 Calibration

There are two kinds of scanner calibration, the first is the geometric calibration (absolute and relative positioning of scanner components) and the second is the radiometric calibration (light detection and measurement). Inadequate calibration of either type will markedly lower the performance of the scanner. Both calibration procedures are undertaken from within the Mephisto Extreme software 'Calibration' tab and are now described.

3.5.1 Geometric calibration

Geometric calibration determines the physical set-up parameters of the 3D-scanner - for example the position of the camera and the projector relative to each other (Figure 3.6).



Figure 3.6 Set up for the geometric calibration

The steps to geometrically calibrate the system are as follows:

Step 1: Orientate the flat calibration board with its checker pattern towards the 3D scanner and stabilise its position (absolute position is not critical), and then click “Grab calibration frames”. The system will then project the necessary patterns and grab the frames. Captured images appear in the calibration window (lower right of Figure 3.7) and each captured image is processed for suitability of use according to four metrics of acquisition, each of which will flag green or red according to the quality of the capture. Ideally only rows which are exclusively green are useful, provided the number of checker corners exceeds 14 as indicated by the numerical values adjacent to each flag. Rows which either contain red flags or low corner detection are best excluded and can be deselected from the calibration trial (check box adjacent to each frame number acquisition).

Step 2: Repeat Step 1 five times but adjust the calibration board each time to a different orientation angle.

Step 3: With only the successful trials selected (ticks in check box) click ‘Calibrate System’ to complete the geometric calibration.

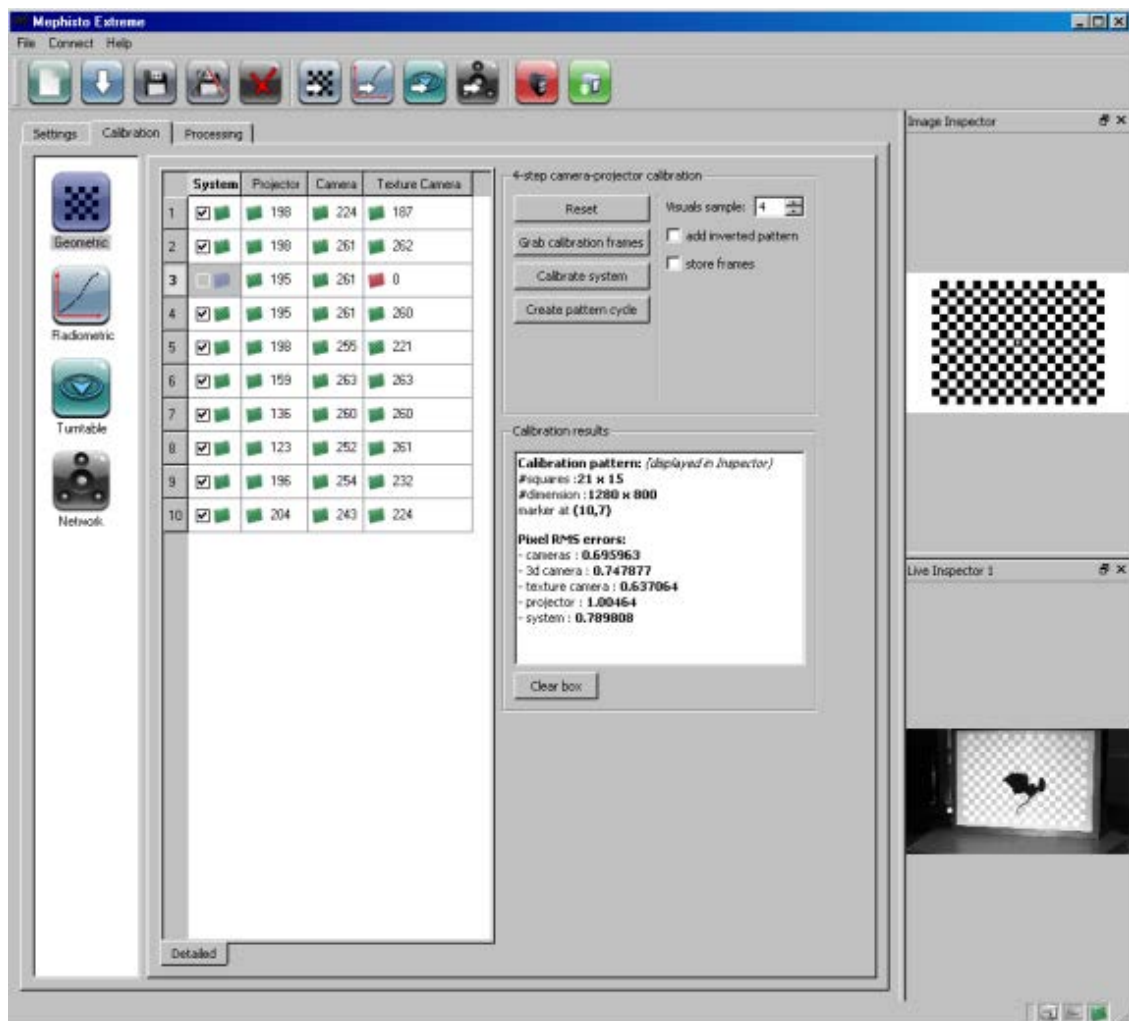


Figure 3.7: Geometric calibration results

After a time interval (may be a few minutes) the calibration results appear on a small table (Figure 3.7). A very successful calibration result is when the error for the cameras and the projector is around 0.25 Pixel RMS errors, however this level of calibration

quality was never achieved for projects for the current work which were more typically 0.7-0.8 and gave medium quality acceptable scans (Table 3.1).

Initial geometric calibration trials were very unsuccessful. For a poor geometric calibration even if the subsequent radiometric calibration was successful the acquired scans of objects would be of unworkable quality. A number of configuration tests were undertaken to identify those conditions that optimise the quality of the calibration and reduce the RMS values to acceptable values, with comparative results illustrated in Table 3.1.

Table 3.1 Comparison of two result sets for low quality versus high/medium quality geometrical calibration runs using; calibration pattern = 21x15 squares, dimension = 1280x800. Numbers are pixel root mean square errors.		
	Low quality calibration	High/medium quality calibration
Cameras:	21.2677	0.695963
3D camera:	17.8074	0.747877
Texture camera:	23.3759	0.637064
Projector:	12.4622	1.00464
System:	14.8553	0.789808

This work established that a lower (acceptable) geometric calibration error can be achieved by:

- a) Ensuring the surface of the calibration board is as flat as possible (no curvature).

- b) Ambient light must be kept to a minimum. The early room location of the scanner required use of black out curtains whilst a later location was in a windowless room.
- c) The captured checker pattern must fill the field of view as much as possible. This was achieved by zooming in to the calibration board so that the checker pattern was exclusively visible.

3.5.2 Radiometric calibration

Radiometric calibration is the normalisation of reflected light intensities from the light source (projector) as detected by the camera sensor over a scale from 0 (black) to 255 (white). This calibration must take place in a very dark room and it does not have to be repeated frequently (typically once per month, with the saved radiometric data being preloaded when starting a new project). Under blacked out conditions the procedure for radiometric calibration is as follows:

Step 1: An A4 piece of clean white paper is placed on the calibration board ensuring that the centre of camera focus is near the centre of the paper (Figure 3.8 left).

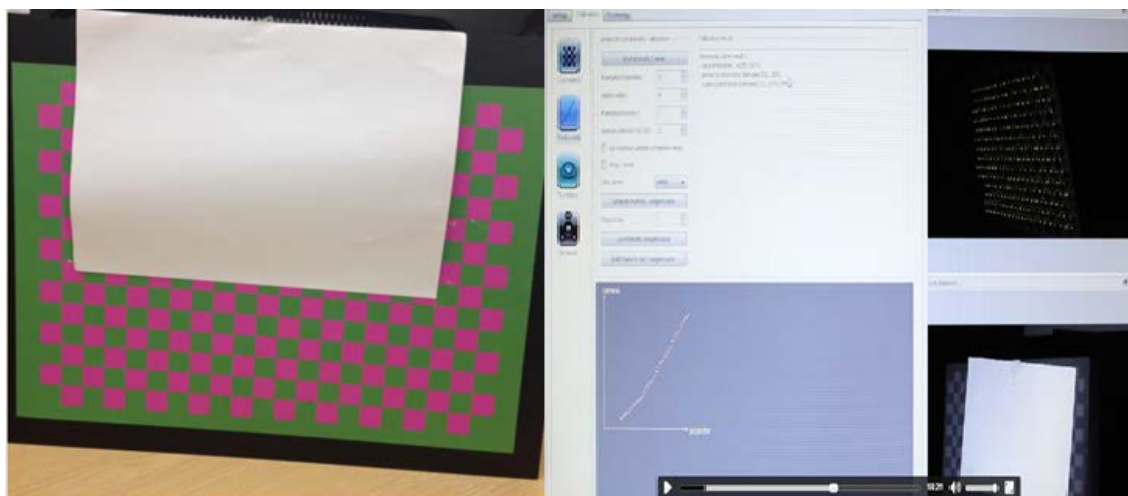


Figure 3.8: How to perform radiometric calibration

Step 2: The camera field of view will display on the computer screen; ensure the balance of white and black is approximately 50%.

Step 3: Click on 'Grab intensity frames'. Using the default settings, 52 different intensities will be captured and these values will be displayed as white dots on a camera vs projector X-Y graph on the computer screen interface. It is particularly important that the camera is not moved during this process otherwise the calibration will be invalidated (Figure 3.8 right).

Step 4: Once the frames have been grabbed, click 'Compute intensity compensation'. This will fit a polynomial to the sampled data expressed as a red curve overlaid on the white dots.

Step 5: 'Use intensity compensation' is then clicked to finalise radiometric calibration. Summary statistics are produced, the most important being the response curve results for used intensities. Ideally this value should lie between 60-65% and indeed all the projects of this work conformed to this radiometric calibration constraint. Figure 3.9 shows a graphic of the final computer screen output window for a project whose radiometric calibration was 63%.

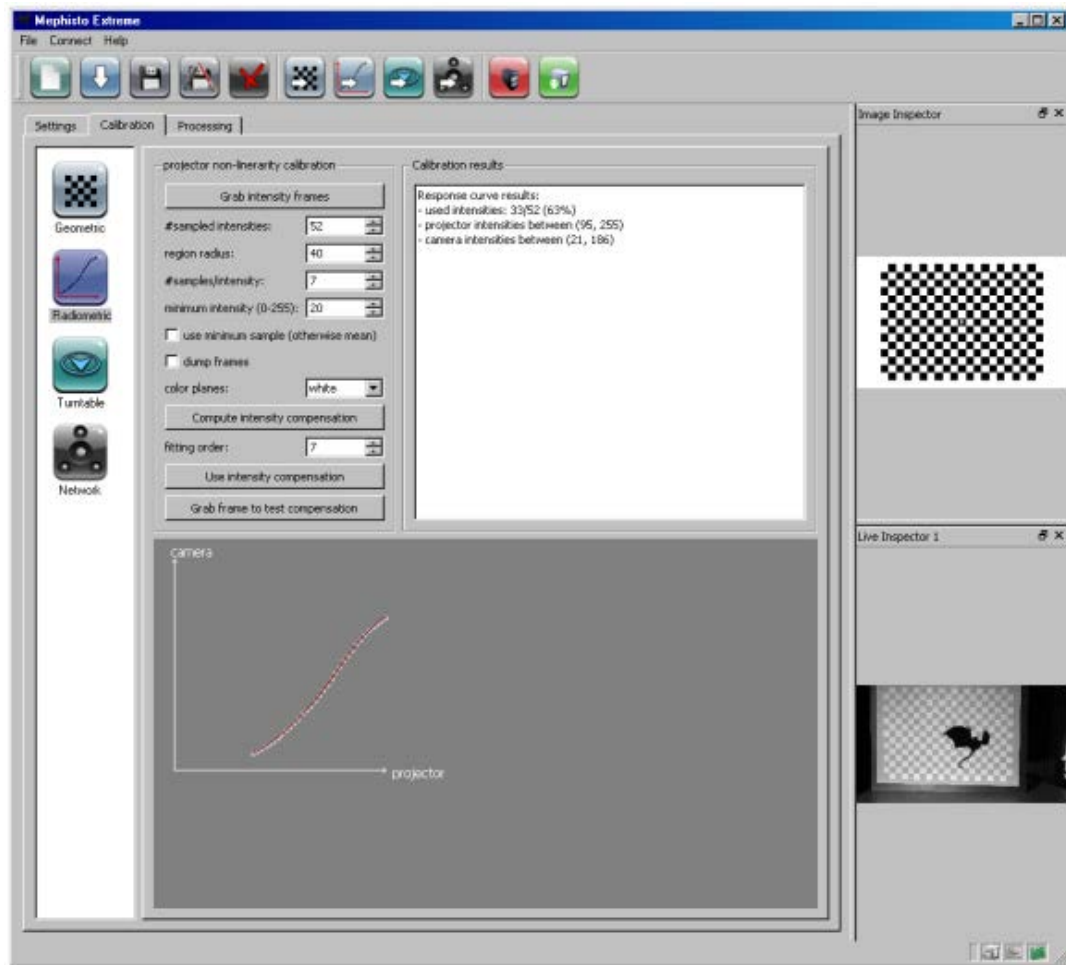


Figure 3.9: Radiometric calibration results

3.6 Calibration verification and the 3D-scanning of objects

Once the scanner calibration is achieved to a desired level of detail/accuracy, the 3D scanning of objects can commence. Pre-scan capture settings accessed via the 3D Acquisition Mode under the Settings tab (Figure 3.5) must be set as follows;

- Vertical gray-code: 8-bits
- Sub-pixel method: 6x60
- Pattern quality: Standard

The user then clicks 'Create pattern cycle'. This may take a few minutes to complete, at which point the projector flickers.

Objects can now be scanned, i.e.

- i) The target is placed in front of the 3D Mephisto scanner
- ii) Camera focus and exposure is adjusted for optimum visibility
- iii) The 'Scan' button is selected from within the 'Processing' tab.

An initial test scan is always undertaken using the calibration board itself, so in step i) above the target is the calibration board (Figures 3.6 and 3.10) with scanning for this object initiated in step iii) (Figure 3.11). This procedure might take a few seconds to complete and then the calibration board appears on the screen (Figure 3.12). A 3D model of the scan can now be viewed by clicking on the '3D model' tab at the bottom of the screen. A 3D image of the calibration board appears and the user can rotate it in every angle (Figure 3.13).

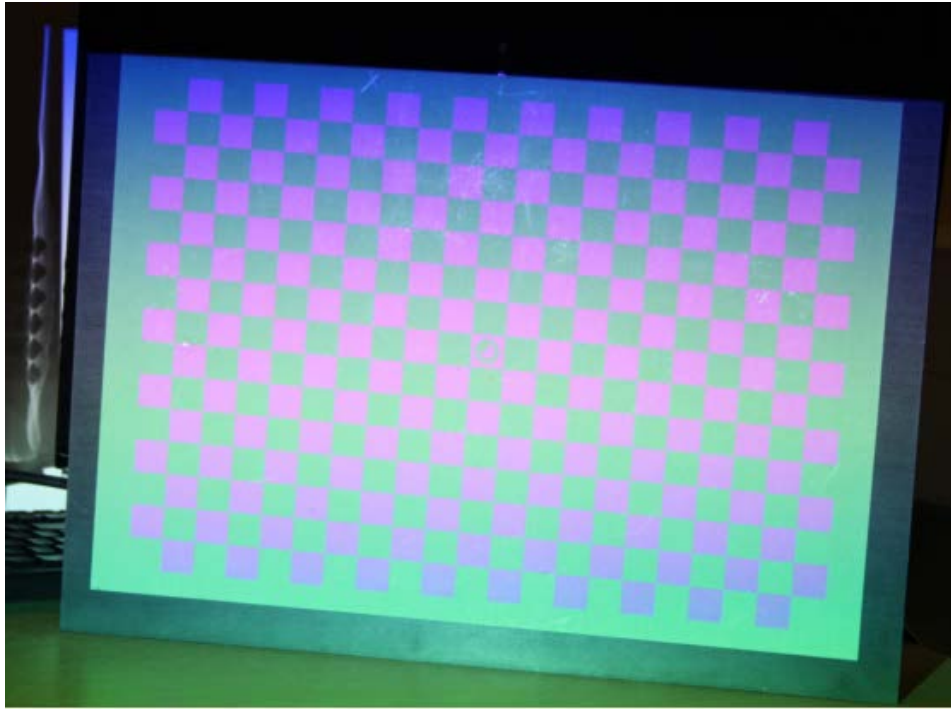


Figure 3.10: The calibration board while scanning

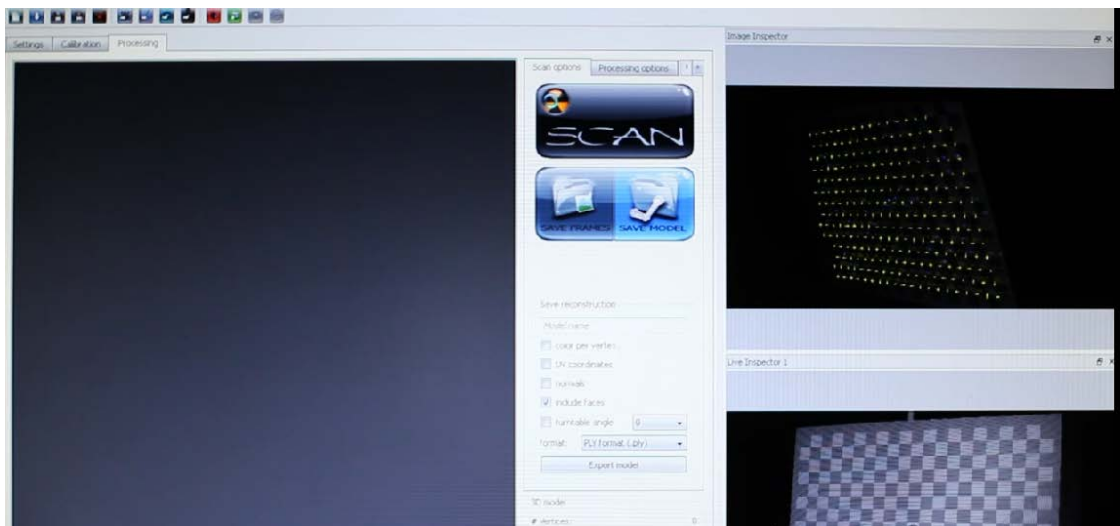


Figure 3.11: The processing tab

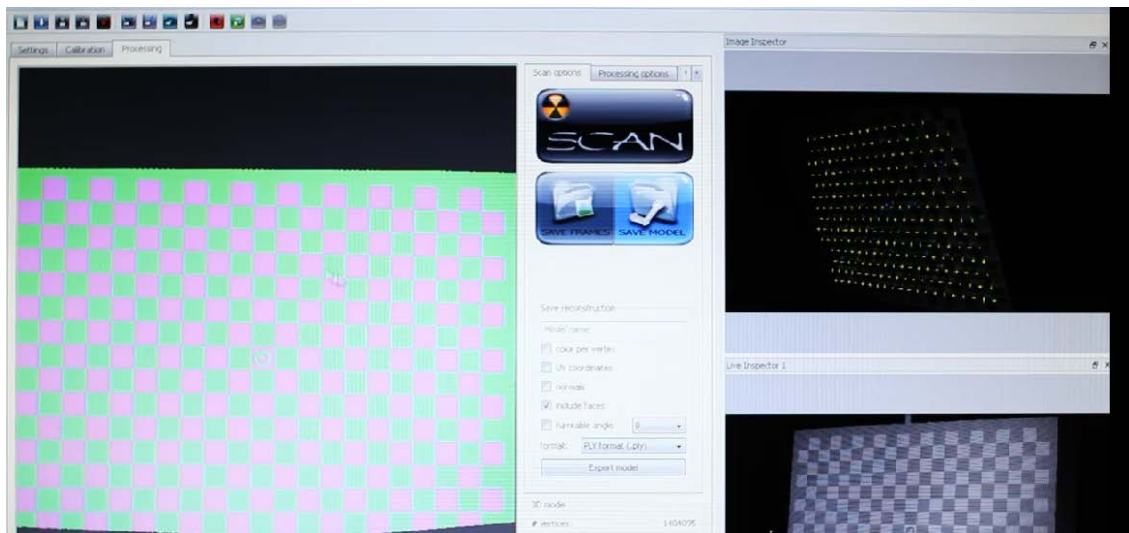


Figure 3.12: Completed scan of calibration board

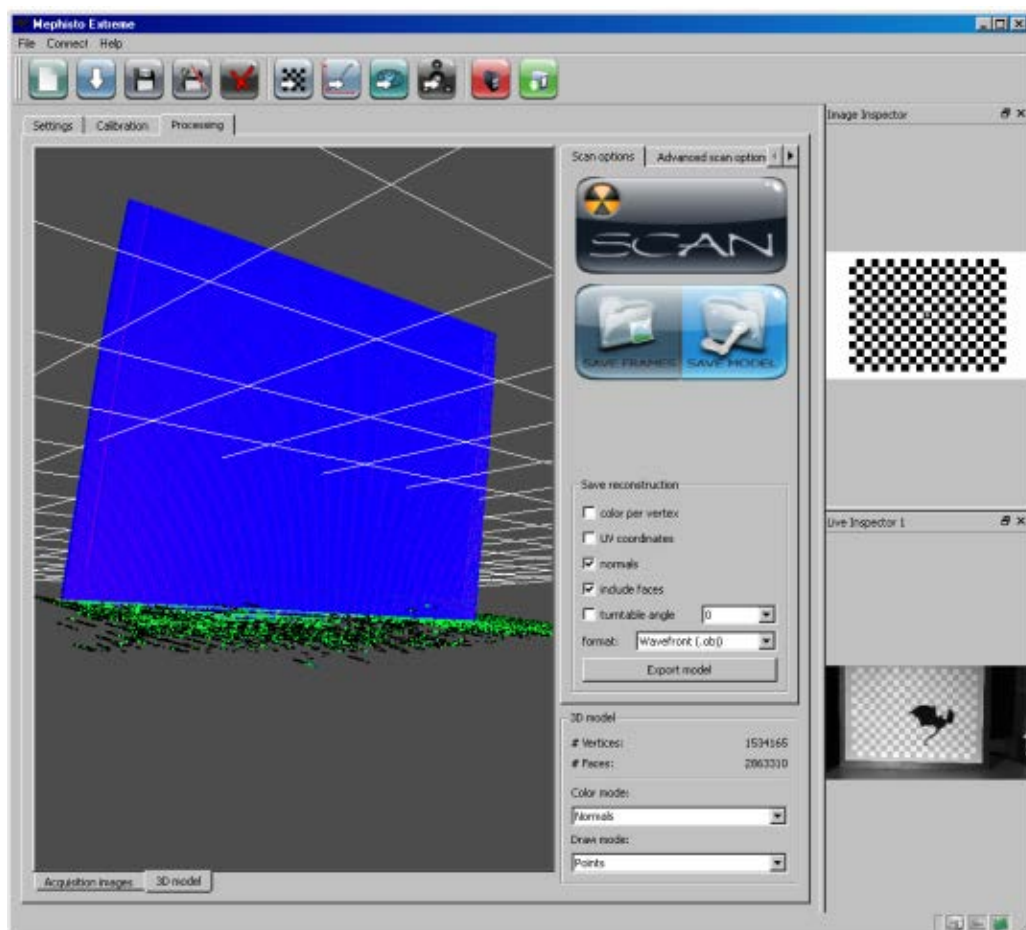


Figure 3.13: 3D model of the calibration board

The 3D model can then be saved and exported in different file formats, and the user can select the details they want to have (normals, UV coordinates, faces, etc). At the 'Color mode' drop down box 'Normals' is probably the best way to display the data as it includes any irregularities of the model.

This completes the scanning process. The user can now repeat steps i) – iii) of this section using another item as the target instead of the calibration board. Note that all the components of the 3D scanner must remain stable at all times, in the same position as during the calibration process. If the unit is moved aggressively, then the calibration should be repeated.

3.7 3D scanning of objects

A number of objects were scanned using the Mephisto 3D scanner. Figure 3.14 shows a figurine whose initial scan was of poor quality, but with appropriate (improved) calibration and configuration was able to generate a highly realistic 3D model of the figurine.

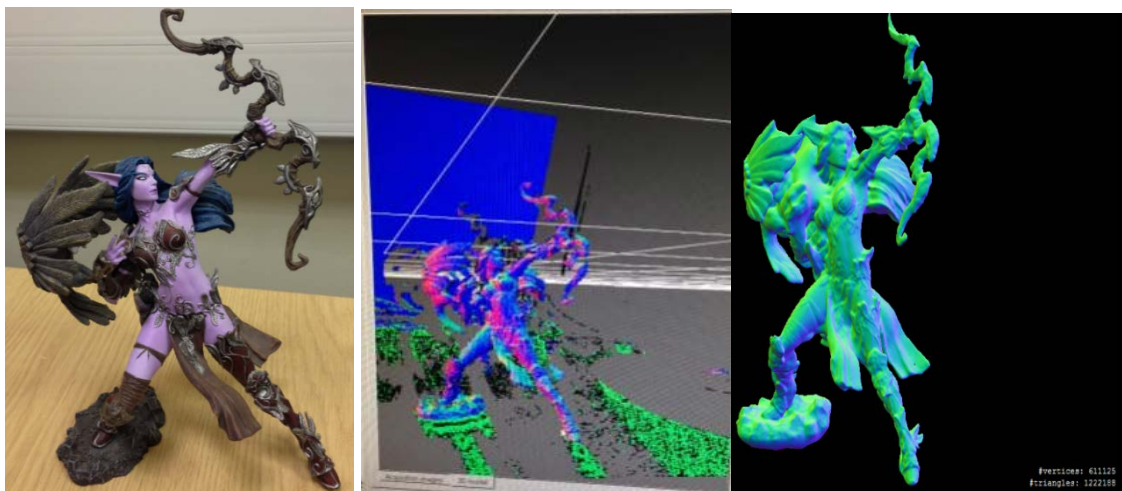


Figure 3.14: Figurine (left) with poor quality scan (centre) and high quality scan (right)

This work has been published by the present author in the academic refereed literature (Constantinou et. al; 2017).

A complete inventory of scanned objects is listed in Table 3.2

Table 3.2 Inventory of objects scanned by Mephisto 3D scanner

Object	Scans	Notes
White box	55	Basic shape
Rubik's cube	31	Basic shape with colour
Box and cube	24	Object of flat surfaces
Anti-slip sign	5	Object of flat surfaces
White box with foil	24	Effect of reflective surface
Rubik's cube with shiny box	13	Colour and reflectivity effects
Pencil case	3	Object of curved surfaces
Bottle	6	Object of reflective curved surfaces
Headphones	2	Object of curved and flat surfaces
Figurine	15	Complex shape
Heads	6	Live subjects
Facial expressions	47	Female subject (author)

The variety of scanned inanimate objects was intended to underpin a number of potential sub-projects within the current research, including studies of the effects of colour and reflectivity on scan quality and 3D object model reconstruction, but were not followed through for reasons already described (logistical issues and insufficiently formulated research questions). A number of scans of the human head, many for the sole purpose of recording facial expressions were also undertaken and that work is described more extensively in chapter 7.

This chapter provided a detailed description of the Mephisto Extreme 3D-scanner calibration, configuration and operation and how 3D object meshes are acquired and saved to file.

Chapter 4:

Object Modelling using 3D StudioMax

4.1 Object modelling versus object scanning

Software that enables the user to design virtual objects is a standard technique in many aspects of engineering and design, the film industry and the game industry. It is a principal technique for designing character/avatar and other objects as step in developing a computer game. The process can be time consuming (weeks, months) and is complimentary to an alternative approach by which real world objects are scanned using a 3D scanner for subsequent processing and import of the digitised object into a computer game. Unlike the design process for an object that does not yet exist, 3D scanning of course requires some real world object to exist as a pre-requisite for scanning. Given the importance and complimentary use of 3D modelling software to 3D scanning, this chapter reviews an application software package called 3DS Max (www.autodesk.com) a commonly used 3D modelling utility (formerly also known as 3D Studio and 3D Studio Max). Whilst 3DS Max is a professional 3D computer graphics program for making 3D animations, models, games and images, this section is confined to an evaluation of its use as a means of creating objects for subsequent import into a computer game (Bosché, 2010; Kersten and Stallmann, 2012).

4.2 Architectural overview

3DS Max has modelling capabilities and a flexible plugin architecture and can be used on the Microsoft Windows platform. It is frequently used by video game developers, many TV commercial studios and architectural visualisation studios. It is also used for movie effects and movie pre-visualization. For its modelling and animation tools, the latest version of 3ds Max also features shaders (such as ambient occlusion and subsurface scattering), dynamic simulation, particle systems, radiosity, normal map creation and

rendering, global illumination, a customizable user interface, new icons, and its own scripting language (Reinhart and Breton, 2009).

4.3 The 3DS Max user interface

Figure 4.1 shows the main user interface of 3DS Max with the four central ‘Viewports’ and the most important options: the main toolbar, the command panels, the time line, the animation tools, the viewport navigator, the time control and the status bar & prompt line.

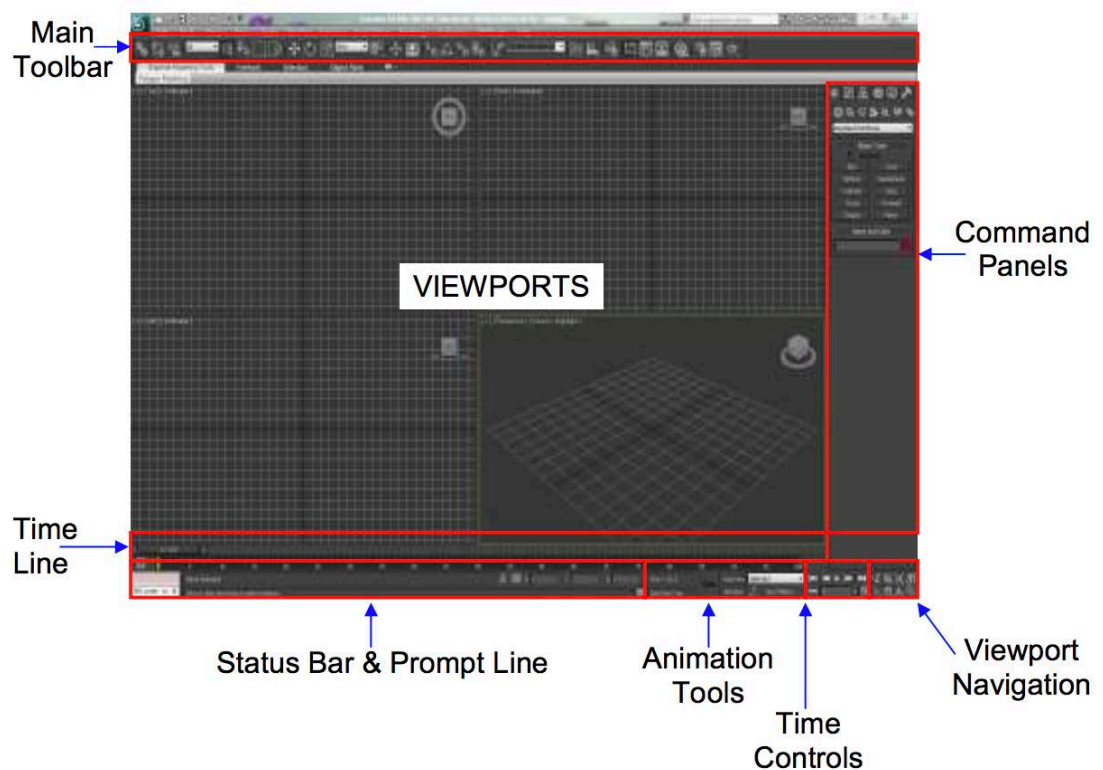


Figure 4.1: The interface of 3ds Max

On the Create Command Panel, the user can create simple 3D shapes by selecting the relevant buttons and then clicking and dragging the mouse cursor in the top viewport. This defines the length and width of the shapes. After releasing the mouse button, drag up and down to adjust the height, and then click once more to set the height. Left clicking

on one of the Viewport descriptors for example displays a menu list for each descriptor, some options for these descriptors which are of interest are:

- PERSPECTIVE, TOP, BOTTOM, LEFT, RIGHT = change view of current Viewport
- SMOOTH + HIGHLIGHTS or WIREFRAME = changes type of display
- OTHER = different ways to view the model
- SHOW GRID = turn HOME grid on/off
- CONFIGURE = change Viewport layout

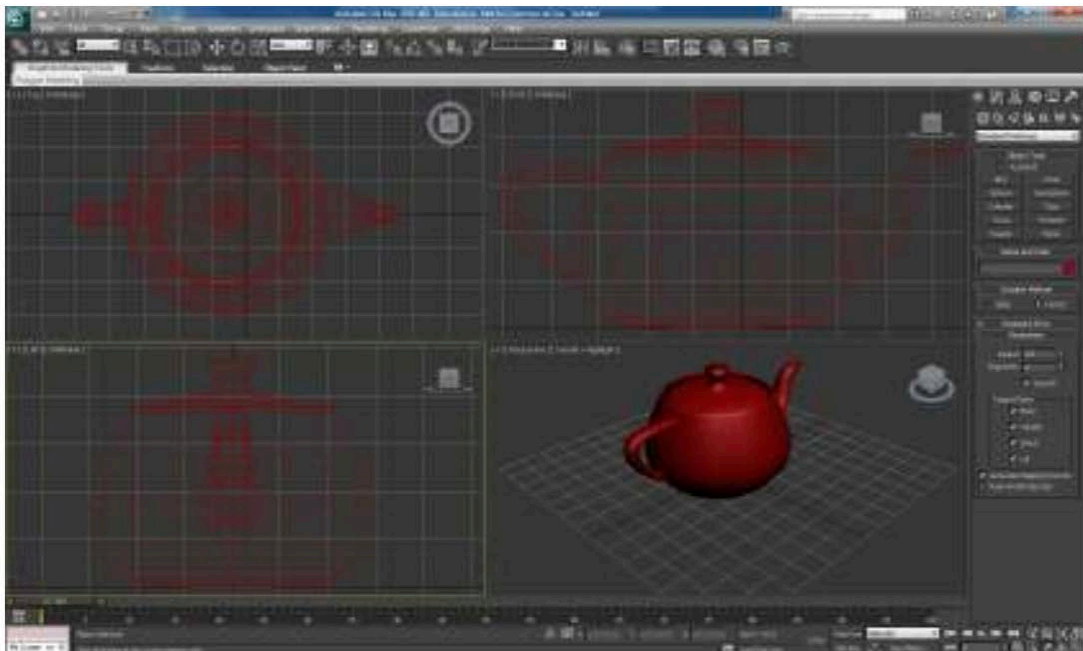


Figure 4.2: viewport manipulation

By default, 3DS Max uses a ‘generic’ unit to represent any required unit e.g. cm, m, km, the user can define the units which are to be used by selecting the customise menu and then Units/ Set Up (Murdock, 2010).

4.4 Creating/adding object primitives

Hover the mouse cursor over the icons immediately under the Create Command Panel icon. This will display the name of the icon. Some of the icons on the Create Command Panel icon which will initially be of interest are:

- **GEOMETRY:** Provides tools to create 3D primitives
- **SHAPES:** 2D shapes (which can be changed into 3D shapes)
- **LIGHTS:** Provides access to various lighting effects
- **CAMERAS:** Provides access to various options which control camera placement and movement
- **HELPERS:** Non-renderable objects (do not appear on the final image) which help with defining objects, shapes and animation
- **SPACE WARPS:** Non-renderable objects which affect objects which are bound to them
- **SYSTEMS:** Special purpose objects Selecting each icon displays a different set of drop-down menus and several options below it.

Clicking and dragging vertically on an empty space on the panel will display further options which may be hidden at the bottom of the panel. For example, the 'NAME AND COLOUR' parameter allows logical names to be given to the objects, whilst selecting the MODIFY COMMAND PANEL will show all the parameters for the object. These options change according to the type of primitive which is selected.

In order to create one object to the side of another object, the user can use the AUTO GRID option in the Create Command Panel.

To do this, the steps below should be followed:

1. Select the object which is to have the new object attached to it
2. Select the type of object you want to attach to the surface using the CREATE COMMAND PANEL
3. Tick the AUTOGRID option, below OBJECT TYPE in the command panel
4. As the cursor is moving over the original shape, the three co-ordinate arrows re-align themselves to the direction of the surface they are currently over.
5. Now click and drag to define a 3D shape as normal.
6. Rather than the object being created on the horizontal HOME grid as usual, the object is created on the surface of the original shape.
7. If the user holds down the ALT key BEFORE they click and drag to create the new shape, the grid which appears on the original objects surface (the FACE grid) will stick to the original object permanently. Any new shapes which are defined from now on will be orientated to this new grid.
8. To return to the original HOME grid, select the FACE grid which is currently active, right click and select ACTIVATE HOME GRID. The FACE grid will remain, but objects will now orientate themselves to the HOME grid. The FACE grid can be re-activated by selecting it, right clicking and selecting 'ACTIVATE GRID'.

4.5 Moving, rotating and scaling objects

When using any of the tools to manipulate the position, orientation and size of a 3D object, a third axis icon called a GIZMO is displayed. By manipulating an axis on the

GIZMO the user can manipulate the selected objects as shown in figure 4.3 (Murdock, 2010).

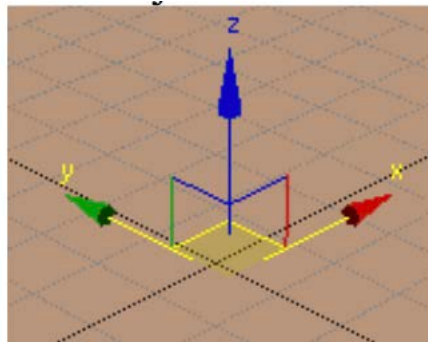


Figure 4.3: Gizmo axis



Selecting objects:

When the last icon in the set of four selection tools is NOT selected, items which are contained within, or which cross the selection area are selected. If selected, items must be completely within the selection area in order to be selected. Use the 'ignore back facing' option to avoid selecting polygons through the model on opposite sides of the model. The 'H' key displays the 'select objects' window, where objects can be selected by their name.



Moving objects: Select an object and then click on the move tool; then click and drag on a gizmo arrow head in a viewport to drag the object along that axis. Click and drag on the edge of 'right angle' box to free-drag the object within the two axes of that view port



Rotating objects: Select the circular icon; the gizmo now looks like a virtual trackball. Circular handles represent the three axes around the trackball, and each viewport provides a handle for a separate axis with the perspective view providing access

to all three handles simultaneously. Clicking and dragging on a colour handle will allow rotation around a single axis. Clicking and dragging in centre of the circle will allow rotation in both axes of that viewport



Scaling objects: Select the square icon; clicking and dragging in centre of gizmo will allow a uniform scaling in all directions. Clicking and dragging on an axis line only will scale in that direction only, whilst clicking and dragging within the diagonal bar between two axes to scale in both those directions simultaneously.



Snapping objects: Select the snap button on the top toolbar; this enables snapping of the cursor to the HOME or FACE grids. To change snap settings select the CUSTOMIZE menu, then GRID & SNAP SETTINGS.

4.6 Editing subsections of a 3D primitive

The ‘Editable Poly’ (short for Editable Polygon) option allows the user to ‘split’ a 3D primitive into subdivisions and modify each division separately.

To achieve this, the user needs to follow these steps:

1. Create a box and select it
2. In the MODIFY panel, use the ‘Length Segs’ ‘Width Segs’ ‘Height Segs’ option boxes to define how the box is to be split up
3. The user may need to left click on the viewport display type in the top left of the viewport (i.e. the words, Perspective or Smooth + Highlights) and then select the Edged Faces option. This will display the new segment lines.
4. Right click on the shape and select ‘Convert to...’ then ‘Editable Poly
5. The shape is now split into different areas.

6. With the object still selected, the MODIFY COMMAND PANEL icon displays an 'Editable Poly' option. Expand this by clicking the + sign next to the 'Editable Poly' text.
7. This list shows all the different elements which can be selected and manipulated, e.g. vertex, polygon.
8. Select POLYGON level and click on the box surface. The user can now select individual subdivision of the box.

Once an editable poly element is selected, (vertex, polygon) the user can rotate, scale and move that component. When using the MOVE tool on polygons, surrounding polygons move out with the moving polygon, and when using the EXTRUDE option on the objects MODIFY COMMAND PANEL, only the selected polygon moves out, expanding the object as it does. Polygons around the edge of the selected polygon remain stationary.

4.7 Copying objects

To copy objects, the user should go to the EDIT | CLONE menu:

- COPY = changes to original shape do not affect new shape
- INSTANCE = Changes to original shape do affect the new shape

Use SHIFT and drag on GIZMO axis with 'Select & Move' tool to create copies.

4.8 Saving files

Object models can be saved by clicking on the relevant Max icon in the top left of the window and then the user select SAVE AS, which saves a copy and changes the current filename to that of the new copy. Clicking the '+' icon saves a copy with '01' appended to the filename, with the number incremented and a new file created every time the icon is clicked, thus supporting incremental backups.

4.9 Import/Export files on 3DS Max

The Import and Export commands on the Application menu allow the user to share 3D geometry with other 3D modelling programs (McHenry and Bajcsy, 2008). 3DS Max can import and export a variety of file formats, as described in Table 4.1:

Import	Export
3D Studio Mesh (3DS) 3D Studio Project (PRJ) 3D Studio Shape (SHP) Adobe Illustrator (AI) AutoCAD (DWG) AutoCAD (DXF) Initial Graphics Exchange Standard (IGES) FiLMBOX (FBX) Lightscape Solution (LS) Stereolithography (STL) VRML (WRL, WRZ)	3D Studio (3DS) Adobe Illustrator (AI) ASC Scene Export (ASE) AutoCAD (DWG) AutoCAD (DXF) FiLMBOX (FBX) Initial Graphics Exchange Standard (IGES) Lightscape Material (ATR) Lightscape Blocks (BLK) Lightscape Parameter (DF) Lightscape Layers (LAY) Lightscape View (VW) Lightscape Preparation File (LP) Stereolithography (STL) VRML97 (WRL)

Table 4.1: Import and Export file formats for 3DS Max

4.10 Example of working with 3DS Max

In order for the researcher to become familiar with the functionality of 3DS Max, an object of appropriate complexity (a jet aircraft) was designed and constructed (Figure 4.4).

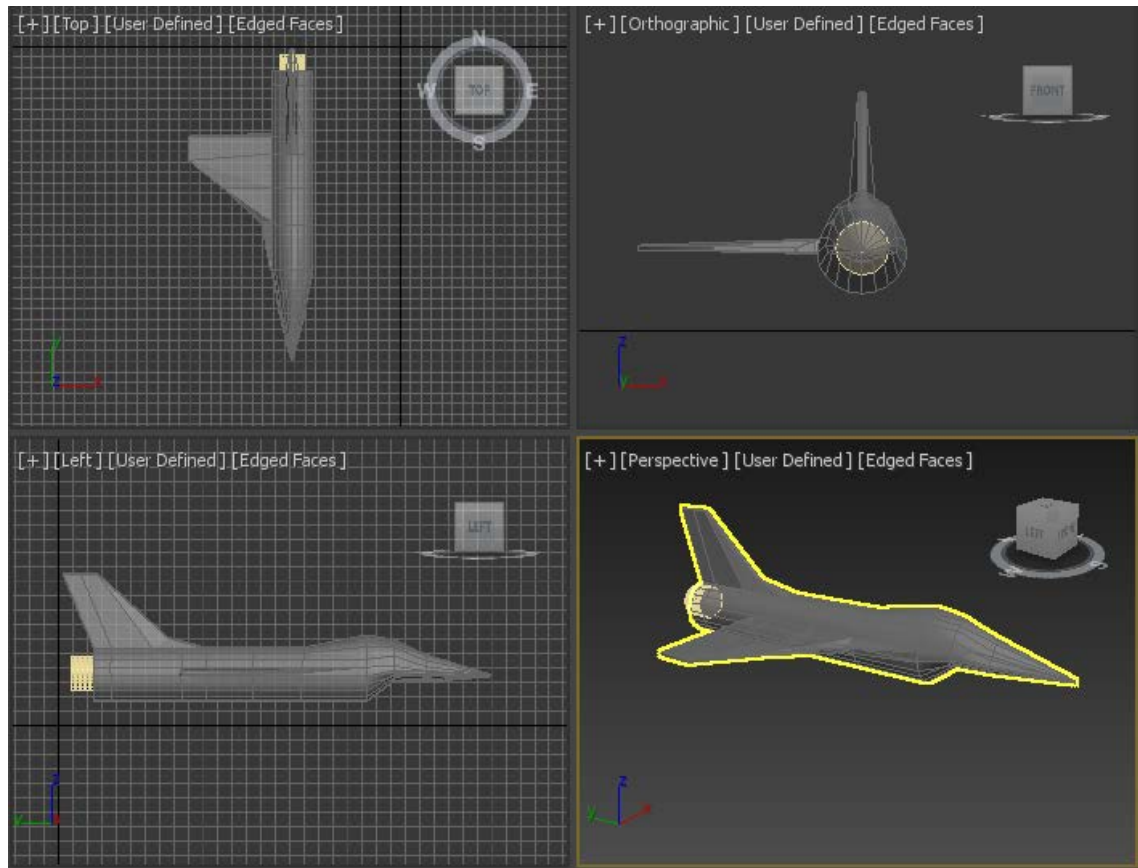


Figure 4.4: Jet aircraft designed in 3DS Max by author

This chapter focused on 3DS Max, a computer graphics modelling environment widely used for commercial and academic purposes to create and edit 3D objects, whether designed or imported.

Chapter 5:

3D objects in Unity3D

5.1 Unity3D for 3D object and game development

Unity3D* is a game development tool that enables the user to develop virtual environments and to populate that environment with objects (known as game assets). The software has functionality to develop 3D objects and in this sense is similar to 3DS Max. However, the game asset building tools are more limited. Nonetheless, one advantage of the Unity game engine is that it supports the full game development process. For this reason it is quite common for 3D objects to be developed in 3DS Max and for those objects to be then imported into Unity. Not only can objects created by 3DS Max be exported and then imported into Unity, but also other objects acquired by other means (e.g. 3D scanning) can also be imported. Use of Unity3D is therefore relevant to the present study of game asset development, both directly from first principles using the functionality of the building tools, and indirectly by import of data file(s) containing shape information from other sources including 3D scanning. This chapter provides an overview of the use of Unity in developing game assets, as well as considering the import methods necessary to access files containing 3D shape information, whether generated by 3DS Max or by 3D scanning. Whilst Unity3D is supported by the Windows and Mac operating systems, the focus of the following outline of use is based on using a MacBook with operating system macOS High Sierra version 10.13.2 (17C88) and the Unity version 2017.3.0f3 Personal.

5.2 Unity3D projects and game development

With Unity3D installed, a user starts by creating a new project, i.e.:

1. Select the Create new project tab

* www.unity3d.com/unity

2. Browse to the desktop and create a new folder where all the Unity projects will be saved

Unity then launches and displays the main interface (Figure 5.1), which by default will be a basic Unity project and from which the user can organise the screen in any way convenient to them.

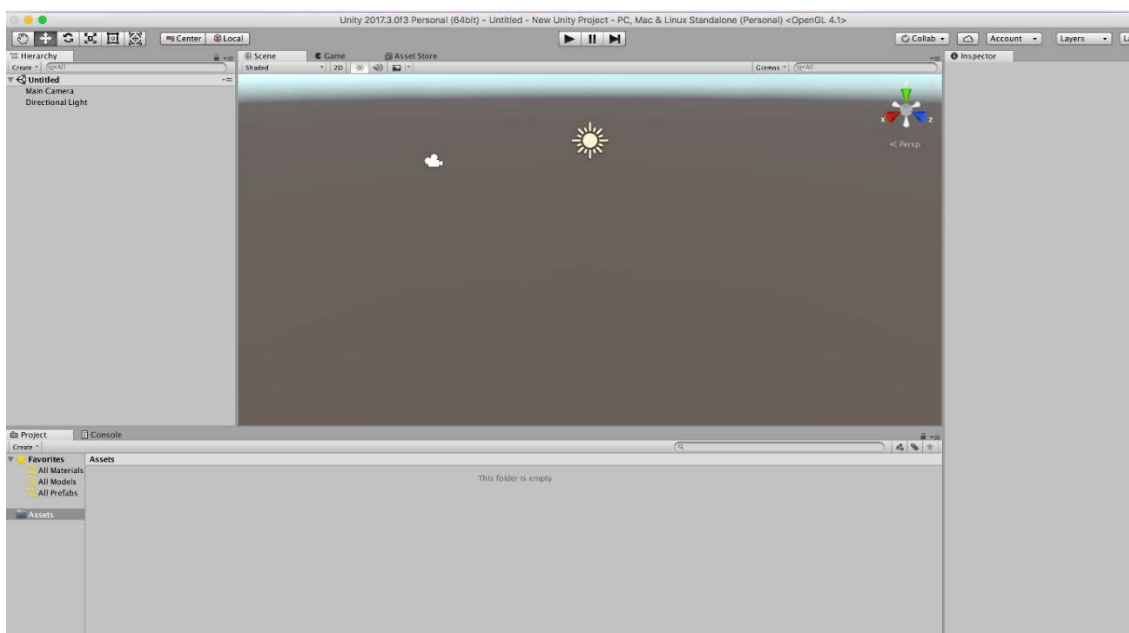


Figure 5.1: Basic Unity project

Unity is a project-based development environment structure in keeping with most other game engine tools and allows its users to have access to a directory where all the assets are stored. The better the user is in organising their assets the easier it will be for them to build their game. The user also has access to Unity packages that allow the user to start their projects with several example files and these are saved as Standard Asset Packages (Creighton, 2010).

5.3 The Unity User Interface (Unity UI)

The Unity User Interface or Unity UI is broken into four basic components (Figure 5.1). The four basic components of the bulk of the UI start at the top left. By default, the Unity UI is laid out to have the hierarchy window at the top left corner. This is where everything that is related to the scene of interest is stored. On the right there is the view port underneath the scene tab where it represents everything that is live in the scene. In the scene there is only the camera and the icon that resembles the sun which is named 'directional light'. Everything that is live in the scene will be populated underneath the hierarchy (Goldenstone, 2011).

The game view is a preview of how everything would look like in a game. It is important to note that this is a test area of looking at how everything comes together as a game view. When or should the user produce an application in Unity the result will be even better than what the game view suggests because game view is a preview as to how everything is being assembled from scene view.

There is a tab named Asset Store where the user can search for free assets or even purchase content from the Unity Asset Store.

On the far right there is the Inspector view where the 'Services' tab is based. Services will demonstrate all the analytics on the games the user may build and anything related to the performance of the game. This is a tab that allows the user to administer all of the parameters of the productivity of the application for anything that may be within the scene view or the hierarchy.

The fourth major area is the Assets view or the project window where the user can look at all of the assets and content directly related to the project.

The Unity UI allows the user to customise layouts into whatever configuration the user wants to work. If the user arranges a layout that they are happy with then the user can save that layout and name it (however they want) and they will then be able to recall it as many times as they wish. Users also have the ability to delete layouts or revert to factory settings or go back to the default layout.

5.4 Navigating Unity

Unity 2017 navigates in similar way as any 3D content creation package (Blackman, 2013). The same principles apply in regards to manipulation and navigating around the view port as well as manipulating objects in the scene (Figure 5.2).

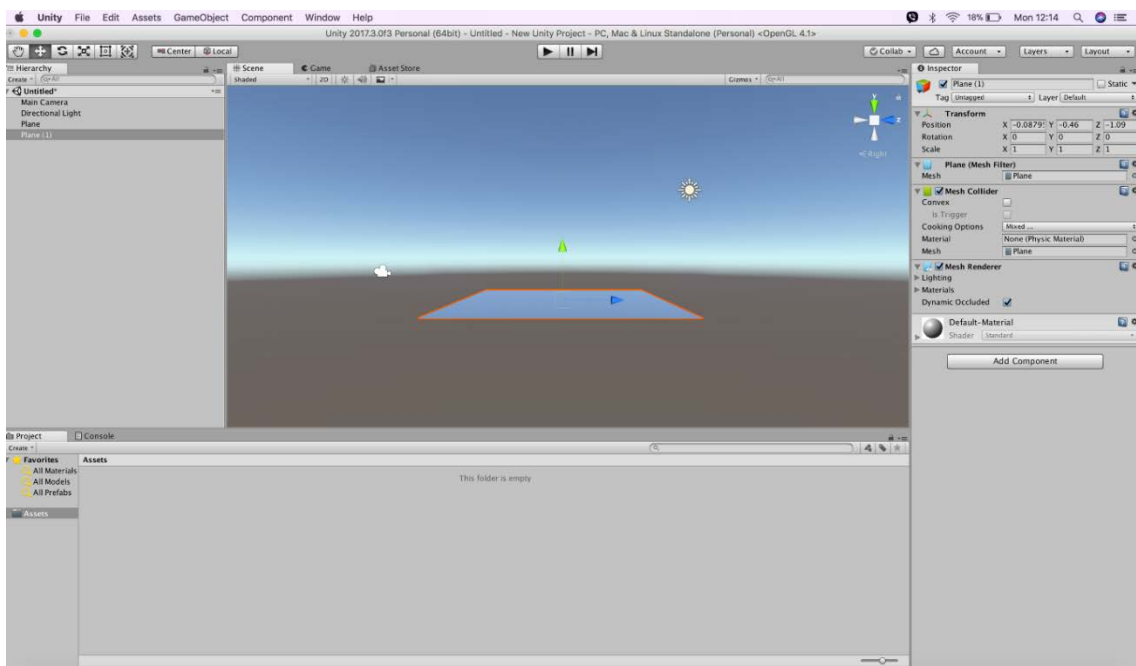


Figure 5.2: Adding a plane in the scene

To add a plane object for example, the user should follow the steps below:

1. Go to the GameObject tab at the top
2. Select 3D object
3. Select plane

This will create a plane in the scene and it will be also added to the hierarchy underneath the chapter name. Also, when the plane is selected, the inspector is visible on the right hand side with all the parameters. Some hotkeys are listed in Table 5.1.

5.5 Game objects and asset creation

Game objects are the most important item in the Unity development environment. Game objects are essential building blocks of any project. They can be found on the top menu Game object (Figure 5.2) and there the user can create anything for the scene: from 3D objects including primitives such as cube, sphere and simple plane, but also basic effects, lights, audio, video, user interface elements and cameras. In the given scene there are already the basic tools that the user will need to build their environment. When the user clicks on the camera the inspector will come up with all the parameters involved. The plane and the camera have in common a name or a label for what the object represents and they also have a transform tab. The best way to describe a game object is that it is a container to hold attributes or parameters. In Unity they are referred to as components that comprise the object (Watkins, 2011).

To create a blank game object, the user has to follow the instructions below:

1. Go to Game Object and click Create empty
2. Add component
3. Click on Rendering
4. Click Light
5. Go back to MyCustomLight

Table 5.1: Mac Unity Hotkeys*

HOLD	+	Key	Function
File			
	Cmd	N	New
	Cmd	O	Open
	Cmd	S	Save
Shift	Cmd	S	Save Scene as
Shift	Cmd	B	Build
	Cmd	B	Build and run
Edit			
	Cmd	Z	Undo
Shift	Cmd	Z	Redo
	Cmd	X	Cut
	Cmd	C	Copy
	Cmd	V	Paste
	Cmd	D	Duplicate
	Shift	Del	Delete
		F	Frame (centre) selection
	Cmd	F	Find
	Cmd	A	Select All
	Cmd	P	Play
Shift	Cmd	P	Pause
Alt	Cmd	P	Step
Assets			
	Cmd	R	Refresh
Game Object			
Shift	Cmd	N	New game object
Alt	Cmd	F	Move to view
Shift	Cmd	F	Align with view
Window			
	Cmd	1	Scene
	Cmd	2	Game
	Cmd	3	Inspector
	Cmd	4	Hierarchy
	Cmd	5	Project
	Cmd	6	Animation
	Cmd	7	Profiler
	Cmd	9	Asset store
	Cmd	0	Asset server
Shift	Cmd	C	Console

* Source: https://docs.unity3d.com/uploads/Main/Unity_HotKeys_Mac.pdf

In order to create an asset, the user has to name the game object and add it in components. By default, anything created under game object, 3D object primitives will place a collision around it. Thus, the game object is a container to store any number of attributes or parameters that the user will need. The mesh renderer component is how the scene that will be rendered in game view. The user has the ability to add to the object whatever they want, for example, effects, lights, materials etc.

5.6 Importing and Exporting Unity assets

When it comes to content creation, there is no shortage of applications to create content for Unity. Unity supports a vast number of 3D formats, Autodesk FBX being one of the more commonly used common file formats (Goldstone, 2009).

When the user creates a Unity Project, they are creating a folder - named after the name of the project - and contained subfolders as shown in Figure 5.3.

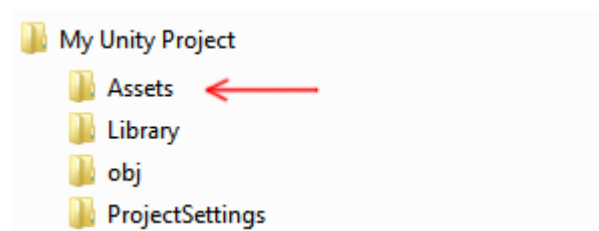


Figure 5.3: The basic file structure of a Unity Project

The **Assets** folder is where the user can save or copy files that they want to use in their project. The contents of the **Project Window** in Unity shows the items in the Assets folder. So if the user saves or copies a file to their Assets folder, it will be imported and become visible in their Project Window.

Unity will automatically detect files as they are added to Assets folder, or if they are modified. When the user puts any asset into their Assets folder, they will see the asset appear in their Project View.

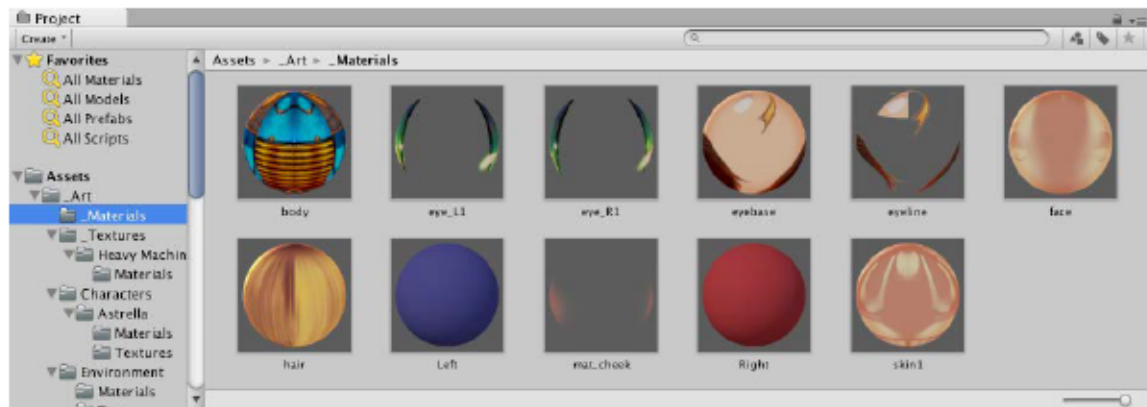


Figure 5.4: The Project Window shows assets that have been imported into the project

If a file is dragged into Unity's Project Window from a computer (e.g. from the Finder on Mac, or from Explorer on Windows), it will be *copied* into the Assets folder, and will appear in the Project window (Figure 5.4).

The items shown in the Project window represent (in most cases) actual files on the computer, and if they are deleted within Unity, they are deleted from the computer too. Figure 5.5 shows an example of a few files and folders inside the Assets folder of a Unity project. The user can create as many folders as they like and use them to organise their Assets.

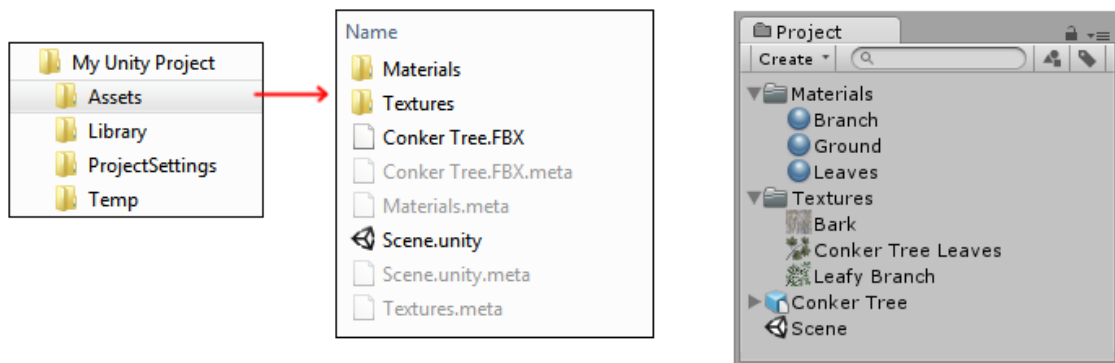


Figure 5.5: The relationship between the Assets Folder in a Unity Project on a computer, and the Project Window within Unity

Of particular relevance to the current work was to ensure/devise a procedure that allowed 3D objects created and/or modified in 3DS Max to be exported as a file from that application and then imported into Unity. The 3D objects created in 3DS Max can be saved in the same way described above directly into the **Project** or they can be exported into Unity using the **Autodesk .FBX** or other generic formats.

Unity imports meshes from 3DS Max. Saving a Max file or exporting a generic 3D file type each has advantages and disadvantages. On import or export to and from Unity the application can transfer the following information:

1. All nodes with position, rotation and scale. Pivot points and Names are also imported.
2. Meshes with vertex colours, normals and one or two UV sets.
3. Materials with diffuse texture and colour. Multiple materials per mesh.
4. Animations.
5. Bone based animations.

5.7 Examples of working with Unity 3D

In order for this researcher to become familiar with the Unity 3D game building tools a virtual 3D environment was developed containing two kinds of assets (3D objects), namely trees and aeroplanes. The trees were located on a plain grey surface and the aeroplanes against a sky blue atmospheric background whose boundary represents the horizon (Figure 5.6).

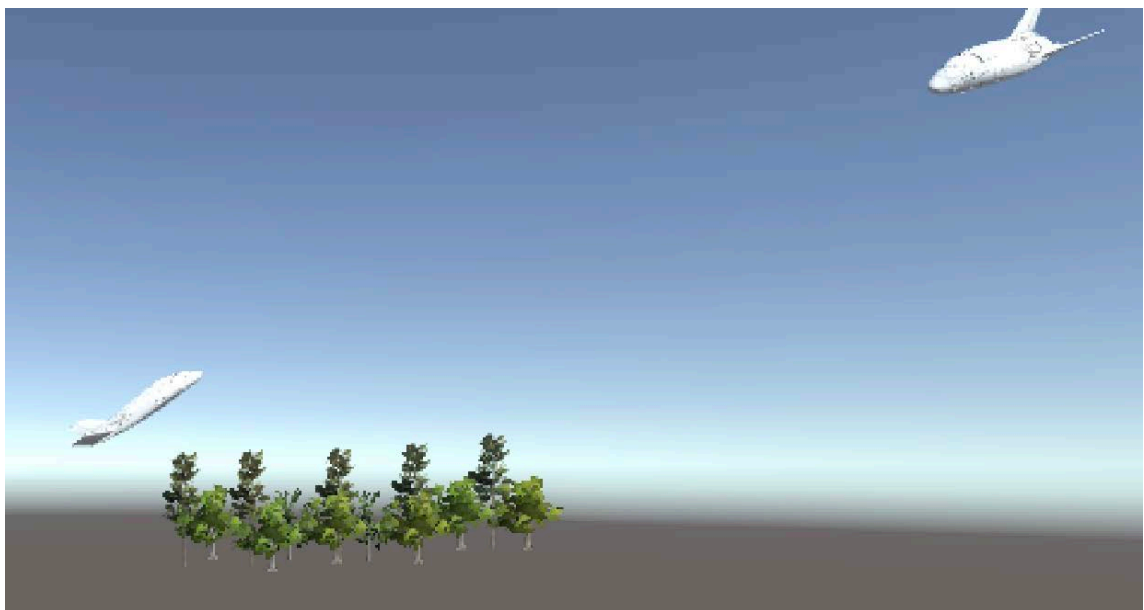


Figure 5.6: Screen capture of Unity 3D game scenario in development

This chapter described the manipulation of 3D objects in Unity3D, a common computer game-building software development environment and to ensure a procedure that allowed 3D objects created and/or modified in 3DS Max to be exported as a file from that application and then imported into Unity.

Chapter 6:

Developing an OBJ file processing tool

6.1 3D Object data portability between applications

Whilst 3D data for an object can be acquired directly by use of a scanning device such as the Mephisto Extreme, 3D data can also be constructed (i.e. virtually designed) for an object using software such as 3DS Max. In order to allow software such as 3DS Max to process 3D data acquired by the Mephisto Extreme, there is clearly a requirement to ensure relevant 3D data can be exported by the scanner software in a file format that can be read by 3DS Max. The significance of working with a file format common to both applications is that it can allow intermediate processing of the 3D data by other bespoke applications. In this section such an application is presented developed in C++ which generates statistical data on the 3D object in question.

Initially for this work a simple 3D object was constructed in 3DS Max (a cube) and exported as an ASE file (a common 3DS Max file format). The ASE files are text files (ASCII files) and they can be read by many applications including a user-written program for the purpose of implementing 3D data processing algorithms. However, there is very little documentation on how to read and interpret ASE files, and importing the ASE files back into 3DS Max proved difficult. For this reason the use of ASE files was abandoned and instead the OBJ file type was adopted as the principal input/output file used in this research. Like ASE files, they are ASCII-based and are easily readable by a text editor. They are readily compatible with 3DS Max for input/output purposes and unlike ASE files there is good documentation available on the type structure and the types of data and its format. Figure 6.1 shows illustratively a comparison of the content of the two file formats, each containing the same information for a cube.

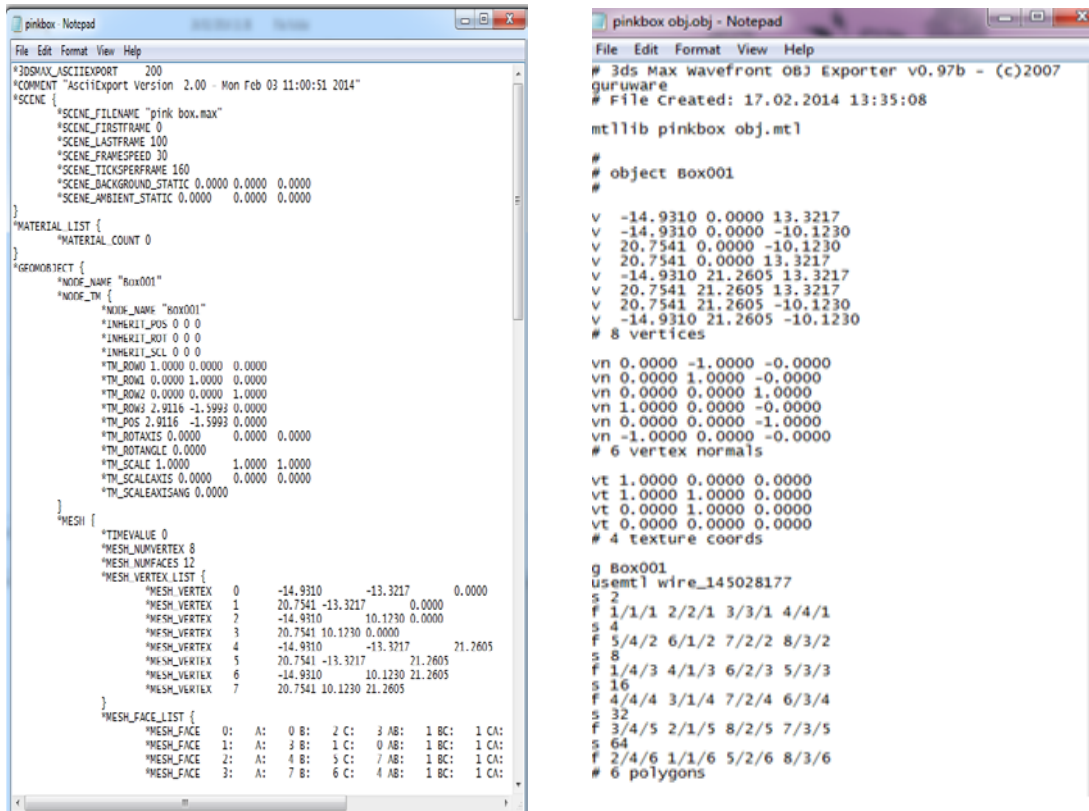


Figure 6.1: Left: ASE file format for a cube; Right: OBJ file format for the same cube.

6.2 Features of Wavefront OBJ files

Wavefront OBJ (object) files store geometric objects composed of lines, polygons, and free-form curves and surfaces. The filenames are often given the extension ".obj" and the format is the principal file associated with Wavefront's Advanced Visualizer 3D modelling software application. The format allows many applications to transfer geometric data (3D, 2D) back and forth between various applications besides the Advanced Visualizer itself, such as 3DS Max, Mephisto Extreme and MatLab (Mathworks, 2011). Whilst Wavefront Object files can be in ASCII format (.obj) or binary format (.mod) the current work concentrates on the ASCII format for object files as described by Reddy (2007).

6.3 OBJ file structure

The first character of each line in an obj file specifies the type of command (apart from the '#' symbol indicating a comment). The first character is followed by various arguments and explained as follows (square brackets indicates an optional argument).

v x y z

The character 'v' indicates the vertex command specified by its three coordinates x, y and z. By default the first listed vertex in the file will be assigned a reference number of 1, and all subsequent vertices sequentially numbered according to the order they are stored in the file (the second vertex listed in the file for example will be reference number 2. These vertex commands only specify points in space and say nothing about the geometry.

vt u v [w]

The 'vt' command refers to vertex texture command, followed by its co-ordinates which are floating point values between 0 and 1 and say how to map the texture in 1D, 2D, or 3D.

vn x y z

The 'vn' command is 'vertex normal' and specifies a normal vector of x, y and z coordinates.

The vertex data is the most important kind of geometric element recorded in OBJ files describing 3D objects created for development purposes in the present work. The vertex data is represented by four vertex lists in an OBJ file; one for each type of vertex coordinates. A right-hand coordinate system is used to specify the coordinate locations. Table 6.1 shows a portion of an .obj file that contains the four types of vertex information.

Table 6.1: Sample of obj file content listing vertex information by type

v	-5.000000	5.000000	0.000000
v	-5.000000	-5.000000	0.000000
v	5.000000	-5.000000	0.000000
v	5.000000	5.000000	0.000000
vt	-5.000000	5.000000	0.000000
vt	-5.000000	-5.000000	0.000000
vt	5.000000	-5.000000	0.000000
vt	5.000000	5.000000	0.000000
vn	0.000000	0.000000	1.000000
vn	0.000000	0.000000	1.000000
vn	0.000000	0.000000	1.000000
vn	0.000000	0.000000	1.000000
vp	0.210000	3.590000	
vp	0.000000	0.000000	
vp	1.000000	0.000000	
vp	0.500000	0.500000	

The vertices are sequentially numbered starting with 1 and provide a reference for use in element statements.

Note that *vt* and *vn* data are often grouped together in use with the 'f' face command as the following describes.

f v1[/vt1]/[vn1] v2[/vt2]/[vn2] v3[/vt3]/[vn3] ...

The face command 'f' enables a group of vertices to be specified which define a polygon. Each vertex may have a triplet of numbers that reference vertex data. These numbers (separated by a forward slash '/') are the reference numbers for a vertex geometry, a vertex texture, and a vertex normal. There may be more than one series of geometric vertex, texture vertex, and vertex normal numbers on a line. For example a line describing a four-sided face element might be:

f 1/1/1 2/2/2 3/3/3 4/4/4

which describes:

f v/vt/vn v/vt/vn v/vt/vn v/vt/vn

If there are only vertices and vertex normals for a face element (no texture vertices), only two slashes (//) are entered. For example, to specify only the vertex and vertex normal reference numbers, it would be displayed as follows:

f 1//1 2//2 3//3 4//4

There is also syntax related to free-form curves and surfaces, but that is of little relevance to the current work and will not be described further.

6.4 The polygonal geometry of a cube

The Mephisto Extreme scanner acquired a number of scans of cubes (Figure 6.2).



Figure 6.2: Positioning of cube on pedestal in preparation for 3D scanning.

The scans were initially undertaken for two purposes;

1. Some tests involved mirroring a selected face with tin foil to determine the reflectivity effects on the construction of the 3D model, with a view to determining what algorithmic mitigation strategies might be applied during post-scanner acquisition processing of the mesh data.
2. Some tests were proposed to scan the (white) cubes in red, green or blue ambient and source light with a view to determining what effects (if any) would occur in the geometry of the acquired 3D data.

In both cases these research topics were abandoned due to difficulty in defining the research questions precisely, the time constraints involved in multiple enforced physical re-locations of the scanner, and difficulties in re-configuration of the scanner associated with the environmental changes that accompanied equipment re-location. Nevertheless the resultant OBJ file content illustrates use of the digitisation reporting and syntax. Table 6.2 describes a cube that measures two units on each side. Each vertex is shared by three different faces.

Table 6.2: Content of obj file listing vertex data for a cube (see also Figure 6.1)

```
v 0.000000 2.000000 2.000000
v 0.000000 0.000000 2.000000
v 2.000000 0.000000 2.000000
v 2.000000 2.000000 2.000000
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
f 1 2 3 4
f 8 7 6 5
f 4 3 7 8
f 5 1 4 8
f 5 6 2 1
f 2 6 7 3
```

6.5 Developing a C++ OBJ reader/writer

Wavefront OBJ file readers and writers are readily available for download from the internet. As the file format specification is in the public domain, many if not most of these programs are open source and completely free for use. However, most readers incorporate rendering functionality through OpenGL, which is not the solution required for the present work. The aim of this aspect of the research was to develop a C++ program that can read an OBJ file, demonstrate access to the geometrical data by printing the vertex data to screen (hence demonstrating proof-of-principle that the 3D data could be processed in some way e.g. apply simplification algorithms) and export (save) it as a separate OBJ output file. Whilst the original input file data and the output file data will be the same, in principle a processing algorithm of choice could be implemented between reading and saving.

One example of a basic OBJ file reader/ writer/ rendering program is given at <http://www.kixor.net/dev/objloader/> and a download of code similar to this was taken and all reference to graphics capability edited out of the version for this work. All C++ development occurred using Microsoft Visual Studio 2015. The code listing for the program developed for this work is presented in Appendix II.

The edited source code is divided into two files, a header file (WavefrontLoader.h) and the file containing their method/function implementations and main (OBJLoader.cpp) - as is common practice in C++ development.

The header file provides information (definitions) for resources used in a multiple file programs including in this case C++ standard libraries <iostream>, <fstream>, <vector> and <string>, as well as a 'using namespace std' to simply standard library methods.

WavefrontLoader.h also includes three user-defined structs. (Vector, FaceElement and Face) and a user-defined class (WFObj).

Note that the Face struct contains a vector of FaceElement structs, i.e:

```
typedef struct
{
    int v;
    int vt;
    int vn;
} FaceElement;

typedef struct
{
    vector<FaceElement> elems;
} Face;
```

The third struct Vector defines a point in 3D space;

```
typedef struct
{
    float x;
    float y;
    float z;
} Vector;
```

Instances of struct Vector are stored in three containers of C++ type vector, whilst a fourth vector container stores instances of struct Face. These are encapsulated within a class called WFObj:

```
class WFObj
{
    private: // Variables to store object data
        vector<Vector> vertices;
        vector<Vector> normals;
        vector<Vector> textures;
        vector<Face> faces;

        void parseLine(string line);
        void parseVertex(string line);
        void parseNormal(string line);
        void parseTexture(string line);
        void parseFace(string line);
    public:
```

```

        WFObj();
        ~WFObj();
        void performOperations();
        int load(const char *filename);
        void write(ostream &ostream);
};

```

The methods of class WFObj are a mixture of private and public scope. The private methods have the following functionality:

parseLine: parses the lines of the obj file imported to the program.

parseVertex: parses the vertices (V)

parseNormal: parses the normals to the faces (Vn)

parseTexture: parses the texture coordinates (Vt)

parseFace: parses the faces (f)

The WFObj class also includes three public methods:

load: imports the data from an obj file.

write: allows writing of object data either to console using ‘cout’ as parameter or to a file using an ‘ostream’ method.

performOperations: this method was developed by the author and takes the stored geometric data about the 3D object and displays the number of vertices, normals, textures and faces (i.e. 4 numbers), i.e;

```

void WFObj::performOperations()
{
    cout << endl;
    cout << "vertices: " << vertices.size() << endl;
    cout << "normals: " << normals.size() << endl;
    cout << "textures: " << textures.size() << endl;
    cout << "faces:  " << faces.size() << endl;
}

```


For the program developed in this work, the C++ method `main()` declares a static instance of class `WFObj` called `Object`, and then calls its method `load(filename)` to parse an obj file and store the data in the various data structures via calls to `parseLine()`. In the version developed here there is a call to the `performOperations()` method from `main()` which simply utilises the ‘`size()`’ method of the respective vector containers to print directly to screen the number of vertices, normals, textures and faces. Whilst this is useful in its own right to crudely compare shape data, no actual modification of geometrical properties occurs in this implementation, but it does demonstrate the proof-of-principle that code could be written in this method to undertake geometric processing between reading the data from an input file and writing it back out to file through appropriate access to that to that geometric data. The last activity of the `main()` method is to export (write) the object data (potentially modified) to file as a an outputfile ‘`my_output.obj`’.

The main method is shown below;

```
int main()
{
    WFObj Object;

    Object.load("purple sphere.obj");           // load data from file
    Object.write(cout);                          // display data
    Object.performOperations();                  // process data
    std::ofstream ofstream("my_output.obj");    // create output stream
    Object.write(ofstream);                     // write data to obj file
    return 0;
}
```

In the current implementation the filename source obj file has to be hard coded into the program and recompiled, though a change to allow filename choice at runtime would be easy to implement.

For testing purposes, a number of obj files were processed by the program, including ‘`pinkbox.obj`’, whose output is shown in Figure 6.3 - note output from the `performOperations()` method are listed at the end of the output.

```

C:\Windows\system32\cmd.exe
# Vertices:
v -14.931 0 13.3217
v -14.931 0 -10.123
v 20.7541 0 -10.123
v 20.7541 0 13.3217
v -14.931 21.2605 13.3217
v 20.7541 21.2605 13.3217
v 20.7541 21.2605 -10.123
v -14.931 21.2605 -10.123
# Normals:
vn 0 -1 -0
vn 0 1 -0
vn 0 0 1
vn 1 0 -0
vn 0 0 -1
vn -1 0 -0
# Textures:
vt 1 0 0
vt 1 1 0
vt 0 1 0
vt 0 0 0
# Faces:
f 1/1/1 2/2/1 3/3/1 4/4/1
f 5/4/2 6/1/2 7/2/2 8/3/2
f 1/4/3 4/1/3 6/2/3 5/3/3
f 4/4/4 3/1/4 7/2/4 6/3/4
f 3/4/5 2/1/5 8/2/5 7/3/5
f 2/4/6 1/1/6 5/2/6 8/3/6

vertices: 8
normals: 6
textures: 4
faces: 6
Press any key to continue . . .

```

Figure 6.3: Screen capture of 3D data output from file ‘pinkbox.obj’.

For more complicated shapes containing thousands of geometrical elements the code in main() which writes to cout (console output) was suppressed and the program recompiled. The data in Table 6.2 shows some geometrical summary data from the performOperations() method, purely to demonstrate the program will handle shapes of various complexities and size. File ‘pinkbox.obj’ loads in less than a second, whilst ‘whitebox.obj’ took 86s to load (Toshiba Portege R930, 2.5GHz dual core, 4Gb RAM). All source obj files are in the Appendices (disk).

Table 6.2: Comparative summary 3D data for various object files

Obj file name	Vertices	Normals	Textures	Faces	Size on disk (Kb)
---------------	----------	---------	----------	-------	-------------------

pinkbox	8	6	4	6	1
purplesphere	482	482	559	512	64
model	41,731	41,731	0	76,395	5,194
whitebox	83,327	83,327	83,327	15,5534	15,516

6.6 Summary of program development

The OBJLoader program was successfully developed in C++ that can read an OBJ file and export (save) it as a separate OBJ output file. Whilst the original input file data and the output file data are the same, in principle this will allow geometrical processing algorithms to be applied between reading and saving via a method ‘performOperations()’. The read/write ‘proof-of-principle’ aim has been very successfully achieved through use of a bespoke source code which was configured to display summary statistics of the object geometry. Objects of any complexity can be read in, the appropriate data extracted for processing, and that data (potentially processed) then written back to file.

This chapter presented the software development aspects of this research, specifically the development of a program for use between object acquisition/creation and import into a game environment through manipulation of a file reader/writer for the OBJ file format.

Chapter 7:

3D Scanning of the human face

7.1 Facial biometrics and emotional expression

Facial expression analysis is an interesting and challenging problem with important applications in many areas such as human–computer interaction and animation (Fasel and Luetin, 2003; Pantic and Rothkrantz, 2000; Pantic and Rothkrantz, 2003; Tian et al. 2005; Davis L.S., 1996). Current facial expression and emotional interpretation methods

are predominantly based on 2D images which classify the input into one of seven basic categories (happiness, sadness, fear, surprise, anger, disgust and neutral). The facial expression recognition occurs in three major steps: Face detection, Feature extraction and Expression classification. This chapter will firstly review the more important techniques in this process, and then introduce 3D scanning (using the Mephisto 3D scanner) as a means by which more effective expression recognition based on 3D geometrical features of the face might be achieved.

An automated facial expression recognition system consists of the following steps:

- Face detection - the isolation of a face location within an image using appropriate measures (Figure 7.1). Note this is quite distinct from Face recognition, which is the identification of an individual on the basis of the measured features, and is not the subject of the present work.

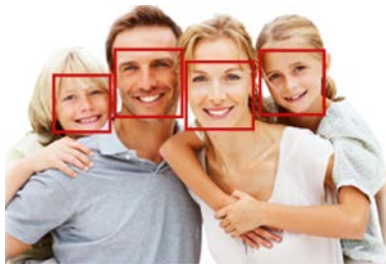


Figure 7.1: Illustrative image of face detection (source: <http://www.divitface.com>)

- Feature extraction - analyses the facial features to eliminate the irrelevant features in the feature selection process. Most techniques are statistical in nature.
- Emotional recognition - the classification step involves identification of the person's emotion from the face in the image. Facial expressions can be used to

identify six basic human emotions; these are anger, fear, happiness, sadness, disgust and surprise (Figure 7.2).

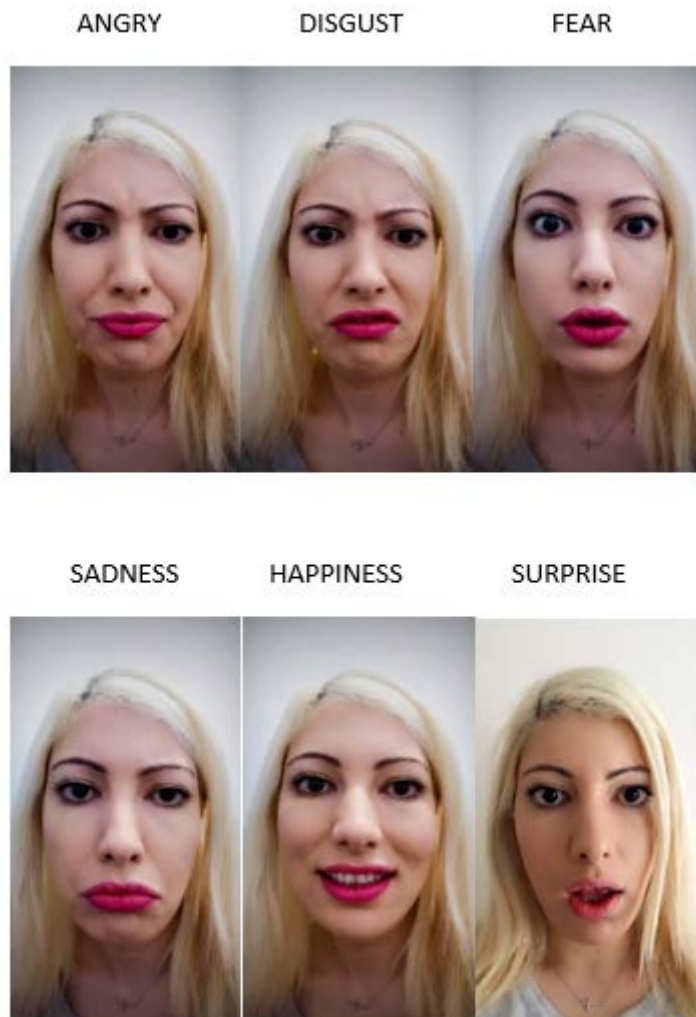


Figure 7.2: Six basic emotions and their typical associated expressions (image of researcher)

There are two common approaches taken to extract facial features: geometric feature-based methods and appearance-based methods (Tian et. al., 2005). Geometric features present the shape and locations of facial components which are extracted to form a feature vector that represents the face geometry. Appearance-based methods are based on image filters such as Gabor wavelets which are applied to either the whole-face or specific face-regions to extract the appearance changes of the face. Of the two approaches, geometric

feature-based methods provide similar or better performance than appearance-based approaches in Action Unit recognition (Valstar et al., 2005). However the geometric feature-based methods usually require accurate and reliable facial feature detection and tracking which is difficult to accommodate in many situations.

7.2 Method of facial expression recognition

Many approaches can be taken when it comes to facial expression and emotional interpretation techniques; these include;

- Principal Component Analysis (PCA) - also known as the Eigen Face Approach. It is one of the most commonly used methods for facial expression interpretation (Sirovich and Kirby, 1987). PCA reduces the dimensionality of the features so that the face indices are retrieved effectively. It uses linear projection and maximizes the projected sample scattering (Belhumeur et al, 1997). This method has only one varying factor and this is the identity of the person and it is not effective when other factors are varying, for example the view point or light conditions. It generates spatially global feature vectors.
- Linear Discriminant Analysis (LDA) - Fisher's Linear Discriminant (FLD) is more suitable when there are severe variations in the facial expressions and lighting conditions. FLD reduces the scattering of projected sample since it is a class specific method (Belhumeur et al, 1997). When compared to PCA the error rate is significantly reduced and it also generates spatially global feature vectors.
- Hidden Markov models (HMM) - these are a set of statistical models used to characterise properties of signals. They work well for images with variation in different lighting, facial expression, and orientation, and are particularly in speech

recognition and character recognition where the data is one dimensional (Nefian, 1998).

- Independent Component Analyses (ICA) - Independent Component Analysis (ICA) generates a statistically independent basis vector which is necessary for effective facial expression recognition (Draper et al, 2003). The biggest advantage of this method is that the average recognition rate is improved but on the other hand it requires more resources and it is more expensive than PCA.

A range of other techniques have been used which mostly either extend the principles of the above and/or are a hybrid of techniques. Briefly, these are Two-Dimensional Principal Component Analyses (Yang et al, 2004), the Global Eigen Approach using Colour Images (Torres et al, 1999), the Sub-Pattern Extended 2D-Principal Component Analysis (Chen et al, 2009), Multilinear Image Analysis (Thomas et al, 2008), Colour Sub-Space Linear Discriminant Analysis (Vasilescu and Terzopoulos, 2002), the 2D Gabor Filter Bank (Barbu and Gabor, 2010), and finally the Local Gabor Binary Pattern (Moore and Bowden, 2011). The most recent techniques have focused on a refinement of the Gabor Filtering technique (Saurav et al, 2015), and on the use of Artificial Neural Networks (Deepthi et al, 2013) utilising databases containing extensive libraries of 2D face images to which these techniques can be evaluated (Jizheng et al, 2013a).

7.3 3D scanning of human facial expressions

One possible application of a 3D scanning facility would be to scan different facial expressions with a view to identifying expression-relevant 3D feature properties. Whilst 3D datasets already exist, the sensitivity of the Mephisto 3D scanner provides a possible opportunity to acquire expression data at an unparalleled degree of resolution. With that

objective in mind 47 3D data sets of the authors face were acquired by the Mephisto scanner, where facial expressions were presented conforming with six basic emotions (Figure 7.3).



Figure 7.3: 3D models of facial expressions acquired by the Mephisto 3D scanner













7.4 Extending 2D facial expression recognition into the 3D domain

It is now recognised that emotion in the human face is communicated by subtle changes in one or a few discrete facial features, such as tightening of the lips in anger or obliquely lowering the lip corners in sadness (Carroll and Russell, 1997). To capture such subtlety of human emotion and paralinguistic communication, automated recognition of fine-grained changes in facial expression is needed. The Facial Action Coding System

(FACS), (Ekman and Friesen, 1978) is a human-observer-based system designed to detect subtle changes in facial features. Viewing videotaped facial behaviour in slow motion, trained observers can manually FACS code all possible facial displays, which are referred to as action units and may occur individually or in combinations. FACS consists of 44 action units. Thirty are anatomically related to contraction of a specific set of facial muscles (Ekman, 1989) and the first 28 are shown in Figure 7.3.

In facial expression analysis, a simplifying assumption is that expressions are singular and begin and end with a neutral position. In reality, a facial expression is very complex especially at the level of action units. Action units may occur in combination or show serial dependence and transitions from action units or combination of actions to another may involve no intervening neutral state. Figure 7.4 illustrates the effect of combining action units to create an expression. For example, an additive combination is smiling (AU 12) with mouth open, which would be coded as AU 12+25, AU 12+26, or AU 12+27 depending on the degree of lip parting and whether and how far the mandible was lowered. Further complications involve the intensity of a facial expression and the difference between a spontaneous as opposed to a deliberate facial expression.

Table 7.3: FACS action units (AU) *

Upper Face Action Units					
AU 1	AU 2	AU 4	AU 5	AU 6	AU 7
					
Inner Brow Raiser	Outer Brow Raiser	Brow Lower	Upper Lid Raiser	Cheek Raiser	Lid Tightener
*AU 41	*AU 42	*AU 43	AU 44	AU 45	AU 46
					
Lid Droop	Slit	Eyes closed	Squint	Blink	Wink








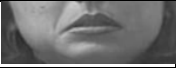






























Lower Face Action Units					
AU 9	AU 10	AU 11	AU 12	AU 13	AU 14
					
Nose Wrinkler	Upper Lip Raiser	Nasolabial Deepener	Lip Corner Puller	Cheek puffer	Dimpler
AU 15	AU 16	AU 17	AU 18	AU 20	AU 22
					
Lip Corner Depressor	Lower Lip Depressor	Chin Raiser	Lip Pucker	Lip Stretcher	Lip Funneler
AU 23	AU 24	*AU 25	*AU 26	*AU 27	AU 28
					
Lip Tightener	Lip Pressor	Lips Part	Jaw Drop	Mouth Stretch	Lip Suck

Table 7.4: Examples of combination of FACS action units**

* Source: <http://www.cs.cmu.edu/~cga/behavior/FEA-Bookchapter.pdf>

** Source: <http://www.cs.cmu.edu/~cga/behavior/FEA-Bookchapter.pdf>

AU 1+2	AU 1+4	AU 4+5	AU 1+2+4	AU 1+2+5
				
AU 1+6	AU 6+7	AU 1+2+5+6+7	AU 23+24	AU 9+17
				
AU 9+25	AU 9+17+23+24	AU 10+17	AU 10+25	AU 10+15+17
				
AU 12+25	AU 12+26	AU 15+17	AU 17+23+24	AU 20+25
				

One of the largest difficulties in this 2D work is that the face orientation relative to the camera may significantly influence the determination of FACs. Face orientation has received deliberate attention in many experiments. For example, the FERET database (Rizvi et al, 1998) includes both frontal and oblique views and several specialised databases have been collected to try to develop methods of face recognition that are invariant to moderate change in face orientation (Vetter, 1995). However in the face expression literature the use of multiple perspectives is rarely used. Most researchers assume that face orientation is limited to in-plane variation (Bartlett et al, 1999) or that out-of-plane rotation is small (Lien et al, 2000, Moses et al, 1995, Rosenblum et al, 1996, Tian et al, 2001).

3D data sets of facial expressions such as those presented in Figure 7.2 could provide a means of progressing techniques in facial expression recognition in two ways:

1. It may be possible to derive new 3D features which are sensitive to facial expressions which are undetectable in 2D images and allow expression recognition. However, this would require the acquisition and evaluation of substantial 3D facial expression mesh data, whilst ignoring the huge amount of 2D work already undertaken and with no guarantee of success at the time of writing. This is probably not the best approach to take.
2. A better approach (in the author's view) would be to take 3D head/expression mesh data and study the effects of face orientation to the camera perhaps, so that an oblique face orientation to the camera can be transformed to an orientation normal to the camera hence allowing a 2D image of the face to be derived, from

which existing 2D techniques for facial expression recognition can be applied. At the present time 2D acquisition of facial images in the community is widespread via CCTV security cameras, but future technology may promote 3D scanning more widely, in which case this work will become increasingly relevant to facial expression recognition and its applications in security and health.

At the time of writing the most recent research includes the following works: a) the FG 2017 Facial Expression Recognition and Analysis challenge (FERA 2017) proposed facial expression recognition systems that address three aspects largely ignored in existing benchmarks: head-pose, expression intensity, and video duration (Valstar et al, 2017); b) A Convolutional Neural Networks based approach for facial expression recognition has been developed reporting an accuracy of 96.76% with real time evaluation (Lopes et al, 2017); c) A facial expression recognition system that learns via deep sparse auto-encoders has provided experimental results which indicate that the presented framework can achieve a high recognition accuracy of 95.79% on the extended Cohn–Kanade (CK+) database for seven facial expressions - this outperforms the other three state-of-the-art methods by as much as 3.17%, 4.09% and 7.41% respectively (Zeng et al, 2018); d) A new face descriptor called the Local Directional Ternary Pattern (LDTP) for facial expression recognition has been developed that efficiently encodes emotion-related features based on image edge patterns (Ryu et al, 2017); e) Lastly, a dynamic framework based on a local Zernike moment and motion history image for facial expression recognition proposes a weighting strategy on a grid for achieving a high recognition rate (Fan and Tjahjedi, 2017).

This chapter considered one specific other application of 3D scanning in its potential impact for improving facial expression recognition. Some facial biometric and emotional expression techniques were reviewed and an experiment presented for extending 2D facial expression recognition into the 3D domain.

Chapter 8:

Conclusion

8.1 Summary of results

The results of this current research are;

1. Identification and procedural description of appropriate graphics modelling software that can process 3D objects from acquisition and/or creation (Mephisto Extreme, 3DS Max), processing (author's OBJ file reader/writer, 3DS Max) and game development (Unity 3D). (Objectives 1,2,3, 4, 5,6,7)
2. Establishment of procedure to achieve successful calibration and configuration of the Mephisto Extreme 3D optical scanner. (Objective 2)
3. Operational development of the scanner enabling acquisition of numerous high quality 3D models from a variety of inanimate target sources. (Objectives 3,4)
4. Software development (in C++) of an OBJ file reader/writer which also demonstrates access to the geometrical data for potential processing purposes. The program provides a link between output from the 3D scanner and input into off-the-shelf modelling and game development software (3DS Max, Unity 3D). (Objective 6)
5. Successful acquisition of a number of 3D scans of the human face presenting a variety of facial expressions. (Objective 7)
6. Identification of the most important parameters in the research literature which focus on 2D facial expression recognition, and propose that 3D expression data could further progress this research. (Objective 7)

8.2 Evaluation of aims

Chapter 1 presented three principal aims to which this research has been directed. The extent to which these aims have been achieved will now be evaluated.

Aim 1 was to *configure, assess and evaluate the Mephisto Extreme 3D Optical scanner as an improved technique for acquiring rapid high resolution 3D object models*. One of the most difficult aspects towards achieving this aim was problems associated with the physical setup, calibration and configuration of the scanner. The Mephisto Extreme is (was) marketed as a portable scanner, delivered in a bespoke stand-alone container which in theory could be moved around as needed (perhaps for example, for use with forensic crime scene capture). However, the author found the portability of the equipment to be severely hampered by its bulk and the need for total blackout conditions during calibration. On two occasions re-location of the equipment caused severe delays in re-establishing operational functionality. In the author's view the Mephisto Extreme is too unwieldy to use for frequent location changes given the calibration/configuration requirements prior to scanning of target objects. Nevertheless, with calibration/configuration established to a satisfactory degree, the routine operational scanning of objects at a high resolution is indeed rapid (a few seconds to a few minutes depending on degree of required detail). One other practical issue that hindered development concerned software updates of both the host computer operating system and the driver software that interfaced with the hardware; such updates were never straightforward with common issues of compatibility needing to be resolved. Furthermore, reflecting on the technology available at the start of this project (2014) with that now available (2018) there have been huge advances in 3D scanning technology – the Microsoft Kinect system for example (widely available on the market in 2014) is capable of acquiring 3D scans which, whilst not of the quality of that yet achievable by the Mephisto Extreme, is just as fast, has an easier calibration/configuration procedure and whose cost is much lower. In short the aim has been met in that this work concludes that the Mephisto Extreme does improve the technique for acquiring rapid, high

resolution 3D scans of objects, but this situation is unlikely to persist for much longer unless there are major improvements in the ease of calibration/configuration accompanying any new advances in high resolution optical scanning technology.

Aim 2 was to *improve 3D object processing techniques that integrate 3D scanning, model construction and game development*. This work shows how use of the Mephisto Extreme scanning software, 3D graphics modelling software (3DS Max) and game development software (Unity3D) can link together 3D object development through use of a modified OBJ file reader/writer developed in C++ by the present author. Whilst the reader/writer does not actually modify geometric values, proof of principle is established in that the software accesses the appropriate data structures, storing the data and reports some statistics to screen. Through use of the modelling techniques described, object data remains accessible from scanning acquisition, to design/modification through to game import. The aim to integrate 3D scanning, model construction and game development has been met, but the degree of improvement over current procedures in that chain has not been quantified.

Aim 3, the final aim, was to *evaluate the use of the optical scanner in the acquisition of facial features and its potential use in facial expression recognition*. This aim is partly achieved in that a number of scans of human facial expressions were successfully undertaken using the Mephisto Extreme. However, work towards a focus on using the 3D data for expression analysis has not been developed. There is substantial existing literature and open source 2D image data sets on this subject and the current work presents a thorough review of that material. Whilst some general ideas as to how the 2D

expression work might be utilised in conjunction with 3D data are outlined, no specific proposal is presented.

8.3 Future work

A number of areas of possible research are referred to in this work but were not developed. However this in turn provides some directions for future research;

1. Development of the OBJ file reader/writer software to modify geometric data.
This would provide a relatively simple means of implementing a particular technique on a surface mesh (e.g. the application of local or global simplification algorithms).
2. Evaluate at what point the reflectivity of a surface renders the output mesh unrecognisable from the original optically scanned object and propose strategies that may correct or mitigate these effects. Related to this is the difficulty that the optical scanning of reflective surfaces produce spurious unrepresentative 3D mesh surfaces, so typically the user would avoid scanning objects with reflective surfaces. Some future work could study this effect with a view to developing strategies to correct for such effects.
3. Perform experiments to identify the relative importance of red, green and blue (RGB) light in determining the quality of the optically scanned object and present results that may allow recommendations for scanner light source configuration in terms of an optimum mix of RGB components.
4. Explore the potential uses of combining the high quality 3D objects acquired from the scanner with 3D printing. For example an object could be scanned, then printed, and the printed object then re-scanned so providing two meshes of the object whose difference could be a measure of quality control.

5. Explore how high resolution 3D scans of facial expressions can be used in facial expression recognition, perhaps through utilising the existing literature and data sets on 2D facial expression recognition.

8.4 Evolving technology

At the start of this current research project the Mephisto Extreme 3D-optical scanner was new state-of-the-art technology for acquiring 3D scans of physical objects. However externally this technology has progressed enormously just within the period over which this current research was undertaken, and new applications are emerging all the time (Droeschel et al, 2017). This is especially seen in medical/biological fields, for example recent new applications include the 3D modelling of joint surface degradation (Gui et al, 2017), white-light 3D body volume and composition scanning (Medina-Inojosa et al, 2017), the aesthetic reconstruction of the congenital condition known as microtia (Ross et al, 2018), and a new method for the recovery and evidential comparison of footwear impressions using a 3D structured light scanning (Thompson and Norris, 2018).

This last chapter presented the main results of this research project, assessed the extent to which the aims have been achieved, and has considered some possibilities for future work.

References

- Adeli H. and Kumar S. (1999) Distributed Computer-Aided Engineering for Analysis, Design, and Visualization, CRC Press, Boca Raton, Florida.
- Alvarez R., Noguera J.-V., Tortosa L. and Zamora A. (2007). A mesh optimization algorithm based on neural networks, *Information Sciences: an International Journal*, Volume 177 Issue 23 December, 2007.
- Armbrust, M., Fox, A., Griffith, R. and Joseph, A. D. (2009). Above the Clouds : A Berkeley View of Cloud Computing. *Science*, 53(UCB/EECS-2009-28), 07-013. Citeseer. doi:10.1145/1721654.1721672.
- Attali, D., Lieutier, A., Salinas, D.(2013) Collapsingrips complexes- 29th European Workshop. Available online at:
<http://www.gipsalab.grenobleinp.fr/~dominique.attali/Publications/2013-socg-rips.pdf>. Last accessed on the 18th of October 2016.
- Baraldi P., Canesi R., Zio E., Seraoui R., and Chevalier R. (2011) Genetic algorithm-based wrapper approach for grouping condition monitoring signals of nuclear power plant components, *Integrated Computer-Aided Engineering*, 18(3), pp. 221-234.
- Barr, A. (1981) "Superquadrics and Angle-Preserving Transformations", *IEEE Computer Graphics and Applications*, vol. 18, pp. 21-30.
- Bemis, S.P., Micklethwaite, S., Turner, D., (2014) Ground-based and UAV-based photogrammetry: A multi-scale, high-resolution mapping tool for structural geology and paleoseismology. *J Struct Geol.* 69:163-178.

- Bernard A.A. (1999) Review of State-of-the-Art Reverse Engineering. Proceedings of TCT Conference; Nottingham, U.K.. October, 1999; pp. 177–188
- Bernardini F. and Rushmeier H. (2002) The 3D model acquisition pipeline, Computer Graphics Forum, 21(2), pp. 149-172.
- Birk M, Zapf M, Balzer M, Ruiter N, Becker J. A comprehensive comparison of GPU- and FPGA-based acceleration of reflection image reconstruction for 3D ultrasound computer tomography. Journal of real-time image processing. 2014;9(1):159-170.
- Blackman, S. (2013) “Beginning 3D Game Development with Unity 4: All-in-one, multi-platform game development” TIA
- Blais F. (2003) A review of 20 years of range sensors development. Proceedings of the Videometrics VII, SPIE IS&T Electronic Imaging; 2003; pp. 62–76.
- Bosché, F. (2010) “Automated recognition of 3D CAD model objects in laser scans and calculation of as-built dimensions for dimensional compliance control in construction”, Advanced engineering informatics 24 (1), pp.107-118.
- Campomanes-Álvarez B.R., Damas S. and Cordon O. (2012) Mesh Simplification for 3D Modeling using Evolutionary MultiObjective Optimization, Proceedings of the 2012 IEEE Congress on Evolutionary Computation, pp.359-366.
- Cagnoni S., Lutton E., and Olague G. (2008) Editorial Introduction to the Special Issue on Evolutionary Computer Vision, Evolutionary Computation 16(4), pp. 437-438.

- Carro-Calvo L., Salcedo-Sanz S., Ortiz-García E. G., and Portilla-Figueras A. (2010) An incremental-encoding evolutionary algorithm for color reduction in images, *Integrated ComputerAided Engineering*, 17(3), pp. 261-269.
- Celano U, Hantschel T, Giammaria G. (2015) Evaluation of the electrical contact area in contact-mode scanning probe microscopy. *J Appl Phys.* 117(21):214305.
- Chabuk T., Reggia J.A., Lohn J., and Linden D. (2012) CausallyGuided Evolutionary Optimization and its Application to Antenna Array Design, *Integrated Computer-Aided Engineering*, 19(2), pp. 111-124.
- Chankong V. and Haimes Y. Y. (1983) *Multiobjective Decision Making Theory and Methodology*, North-Holland, Amsterdam.
- Cignoni P., Callieri M., Corsini M., Dellepiane M., Ganovelli F. & Ranzuglia G., (2008) Meshlab: an opensource mesh processing tool. In: Scarano V., De Chiara R. & Erra U. (Eds) – *Eurographics Italian Chapter Conference*, 2008: 129-136. <http://dx.doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136>
- Cignoni P., Montani C., and Scopigno R. (1998) A comparison of mesh simplification algorithms, *Computers and Graphics*, 22(1), pp. 37-54.
- Coello C.A., Lamont G. B. and Van Veldhuizen D. A. (2007) *Evolutionary Algorithms for Solving Multiobjective Problems*, Springer, Berlin.
- Cohen J., Varshney A., Manocha D., Turk G., Weber H., Agarwal P., Brooks F., and Wright W. (1996) Simplification envelopes, *Computer Graphics (SIGGRAPH '96 Proceedings)*, pp. 119- 128.

- Creighton, R. (2010) “Unity 3D game development by example: A Seat-of-your-pants manual for building fun, groovy little games quickly” A beginner’s guide, Packt Publishing.
- Deb K., Pratap A., Agarwal S., and Meyarivan T. (2002) A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation, 6(2), pp. 182-194.
- Del Riego R., Otero J., and Ranilla J. (2011) A low cost 3D Human Interface Device using GPU-based Optical Flow Algorithms, Integrated Computer-Aided Engineering, 18(4), pp. 391-400.
- Droeschel, D., Schwarz, M., Behnke, S. (2017) Continuous mapping and localization for autonomous navigation in rough terrain using a 3D laser scanner. Robotics and Autonomous Systems. Volume 88. P.104-115.
- E-G Models (2014) Available text online at: <http://www.egmodels.de/formats/index.html>
Last accessed: 7th October 2015.
- Ewelle, RE., Francillette, Y., Mahdi, G. (2013) Computer Games: Level of detail based network adapted synchronization for cloud gaming 2013 - ieeexplore.ieee.org
Available online: <http://ieeexplore.ieee.org/> Last accessed on the 26 January 2015.
- Fan X., Tjahjadi, T. (2017) A dynamic framework based on local Zernike moment and motion history image for facial expression recognition Volume 64, April 2017, Pages 399-406.

- Finney, K. (2013) Book: 3D game programming all in one p.26-39 Available online: http://books.google.co.uk/books/about/3D_Game_Programming_All_in_One.html?id=cknGqaHwPFkC . Last accessed on the 26 January 2015.
- Foster S, Brostoff J. (2006) Digital doppelgangers: Converging technologies and techniques in 3D world modeling, video game design and urban design. In: *Media convergence handbook-vol. 2*. Springer; pp.175-189.
- Fröhlich C, Mettenleiter M. (2004) Terrestrial laser scanning—new perspectives in 3D surveying. International archives of photogrammetry, remote sensing and spatial information sciences. 36(Part 8):W2.
- Fujiwara Y. and Sawai H. (1999) Evolutionary computation applied to mesh optimization of a 3-D facial image, IEEE Transactions Evolutionary Computation, 3(2), pp. 113-123.
- Garland M. and Heckbert P.S. (1997) Simplifying surfaces with color and texture using quadric error metrics, Computer Graphics (SIGGRAPH '97 Proceedings), pp. 209–216.
- Garrido-Jurado S, Muñoz-Salinas R, Madrid-Cuevas FJ, Marn-Jimnez MJ. Simultaneous reconstruction and calibration for multi-view structured light scanning. Journal of Visual Communication and Image Representation. 2016;39:120-131.
- Goldenstone, W. (2011) Unity 3.x Game Development Essentials, Packt Publishing .
- Gross M.H., Staadt O.G., and. Gatti R. (1996) Efficient triangular surface approximations using wavelets and quadtree data structures, IEEE Transactions on Visualization and Computer Graphics, 2(2), pp. 130-144.

- Gu'eziec A. (1995) Surface simplification with variable tolerance. In Proceedings of the Second International Symposium on Medical Robotics and Computer Assisted Surgery, MRCAS '95.
- Gui, Y., Xia, C., Ding, W., Qian, X., Du, S. (2017) A New Method for 3D Modeling of Joint Surface Degradation and Void Space Evolution Under Normal and Shear Loads. *Rock Mech Rock Eng* (2017) 50:2827–2836
- Haemer M.J. and Zyda M.J. (1991) Simplification of objects rendered by polygonal approximation, *Computers and Graphics*, 15(2), pp. 175–184.
- Hamann, B. (2004) A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, p.197–214.
- Hamedi K. (2013). Simplygon: 3D Optimisation Available online: <http://www.simplygon.com/> Accessed: 8th May 2013.
- Hayashida M., Ruan P., and Akutsu T. (2012) A Quadsection Algorithm for Grammar-Based Image Compression, *Integrated Computer-Aided Engineering*, 19(1), pp. 23-38.
- He, T. Hong, L. Varshney, A. and Wang S. (1996). Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics*.
- Hebert D.J. and Kim H.J. (1995) Image encoding with triangulation wavelets, *Proceedings SPIE*, 2569(1), pp. 381-392.
- Heckbert P. and Garland M. (1997) Survey of Polygonal Surface Simplification Algorithms, Technical Report, Carnegie Mellon University, Pittsburg.

- Hinker, P. and Hansen, C. (1993) Geometric optimization. In Gregory M. Nielson and Dan Bergeron, editors, *Proceedings Visualization '93*, pp. 189–195, October 1993
- Hoppe H., DeRose T., Duchamp T., McDonald J., and Stuetzle W. (1993) Mesh optimization. In *Proceedings of SIGGRAPH 93* (Anaheim, California, August 1–6, 1993), *Computer Graphics Proceedings, Annual Conference Series*, pp. 19–26. ACM SIGGRAPH.
- Hou J., Chen, Z., Qin X., and Zhang D. (2011) Automatic image search based on improved feature descriptors and decision tree, *Integrated Computer-Aided Engineering*, 18(2), pp. 167-180.
- Huang H.L., Ho S. Y. (2003) Mesh optimization for surface approximation using an efficient coarse-to-fine evolutionary algorithm, *Pattern Recognition*, 36, pp. 1065-1081.
- Ikeuchi K. and Sato Y. (2001) *Modeling from Reality*, Kluwer Academic Publishers Norwell, MA, USA.
- Jafarkhani R. and Masri S.F. (2011) Finite Element Model Updating Using Evolutionary Strategy for Damage Detection, *Computer-Aided Civil and Infrastructure Engineering*, 26(3), pp. 207- 224.
- Kersten, A. and Stallmann, D. (2012) “Automatic texture mapping of architectural and archaeological 3D models.” *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Volume XXXIX-B5.

- Kim S. and Kim C. (2006) Extracting feature lines from unstructured meshes using a model feature map, *Integrated Computer-Aided Engineering*, 13(2), pp. 101112.
- Kjaer KH, Ottosen C. 3D laser triangulation for plant phenotyping in challenging environments. *Sensors*. 2015;15(6):13533-13547.
- Kus A. (2009) Implementation of 3D optical scanning technology for automotive Applications *Sensors* , 2009, Vol.9(3), p.1967 Available online at: <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3345859/> Last accessed 6th April 2018.
- Kus, A. Unver, E. and Taylor, A. (2009) A comparative study of 3D scanning in engineering, product and transport design and fashion design education. *Computer Applications in Engineering Education*. 17(3):263-271.
- Kuzminsky SC, Gardiner MS. Three-dimensional laser scanning: Potential uses for museum conservation and scientific research. *Journal of Archaeological Science*. 2012;39(8):2744-2751.
- Lavoué G., Chevalier L., Dupont F., (2013) Streaming compressed 3D data on the web using JavaScript and WebGL. In: *Proceedings of the 18th International Conference on 3D Web Technology (Web3D '13)*. ACM, New York, NY, USA: 19-27. <http://dx.doi.org/10.1145/2466533.2466539>.
- Lerma, J.L., Navarro, S., Cabrelles, M., Villaverde, V. (2010) Terrestrial laser scanning and close range photogrammetry for 3D archaeological documentation: The upper palaeolithic cave of parpall as a case study. *Journal of Archaeological Science*.;37(3):499-507.

- Li L., Schemenauer N., Peng X., Zeng Y., Gu P. (2002) A reverse engineering system for rapid manufacturing of complex objects. Robot Comput-Integr. Manuf. 2002;18:53–67.
- Lopes, A. T., De Aguiar, E, De Souza, A.F., Oliviera-Santos T. (2017) Facial expression recognition with Convolutional Neural Networks: Coping with few data and the training sample order. Pattern Recognition Volume 61, January 2017, Pages 610-628
- Luhmann T, Robson S, Kyle S, Boehm J. (2014) Close-range photogrammetry and 3D imaging. Walter de Gruyter
- Luo,J., Hu,G.,Ni, G. (2014) - Computer Animation and Virtual Worlds, 2014 - Wiley Online Library, Dual-space ray casting for height field rendering. Available online: <http://onlinelibrary.wiley.com/doi/10.1002/cav.1531/full>, Last accessed on the 26 January 2015
- Ma, T., Gong, G., Yan, J. (2012) A 3D model simplification algorithm based on edge-collapse, Industrial Informatics (INDIN), 2012 Available on – www.ieeexplore.ieee.org Last accessed on the 26 January 2015
- Marano G.C., Quaranta G., and Monti G. (2011) Modified genetic algorithm for the dynamic identification of structural systems using incomplete measurements, Computer-Aided Civil and Infrastructure Engineering, 26(2), pp. 92-110.
- Mathworks (2011) Available text online at:
<http://www.mathworks.co.uk/matlabcentral/fileexchange/27982-wavefront-object-toolbox> . Accessed: 7th October 2015.

- McHenry, K. and Bajcsy, P. (2008) “An overview of 3D data content, file formats and viewers”, National Center for Supercomputing Applications, Technical Report: isda08-002.
- Medina-Inojosa, J., Somers, V, Jenkins, S., Zundel, J., Johnson, L. Grimes, C, Lopez-Jimenez, F. (2017) Validation of a White-light 3D Body Volume Scanner to Assess Body Composition Obes Open Access. 2017; 3(1): 10.16966/2380-5528.127.
- Milroy M., Weir D.J., Vickers G.W., Bradley C. (1996) Reverse engineering employing a 3-D laser scanner: a case study. *Int. J. Adv. Manuf. Tech.* 1996;12:111–120.
- Murdock, K. (2010) “3Ds Max 2011” Illustrated, Wiley.
- Nedelcu RG, Persson AS. (2016) Scanning accuracy and precision in 4 intraoral scanners: An in vitro comparison based on 3-dimensional analysis. *J Prosthet Dent.* 2014;112(6):1461-1471.
- Okino Computer Graphics. (2013) Wavefront OBJ Importer. Available text online at: http://www.okino.com/conv/imp_wave.htm Last accessed: 7th October 2015.
- Okino Computer Graphics. (2013) Wavefront OBJ Exporter. Available text online at: http://www.okino.com/conv/exp_wave.htm Last accessed: 7th October 2015.
- Pallone, M., Meaney, P.M., Paulsen, K.D. (2016) 3d scanning laser systems and methods for determining surface geometry of an immersed object in a transparent cylindrical glass tank.

- Peipe J., Przybilla H.J. (2005) Modeling The Golden Madonna. Cipa 2005 XX International Symposium; Torino, Italy. 26 September – 01 October 2005.
- Pengdong G., Ameng L., Yongquan L., Jintao H., Nan L., and Wenhua Y. (2008) Adaptative Mesh Simplification Using Vertex Clustering with Topology Preserving, CSSE '08 Proceedings of the 2008 International Conference on Computer Science and Software Engineering, 2, pp. 971-974.
- Plemenos D. and Miaoulis G. (2012) (Eds.): Intelligent Computer Graphics 2011, Studies in Computational Intelligence 374, Springer-Verlag, Berlin, Heidelberg.
- Plemenos D. and Miaoulis G. (2008) Intelligent Techniques for Computer Graphics, D. Plemenos, G. Miaoulis editors, Artificial Intelligence Techniques for Computer Graphics 159, pp. 1-14, Springer-Verlag, Berlin, Heidelberg.
- Pucci B., Marambio A., (2009) Olerdola's Cave, Catalonia: a virtual reality reconstruction from terrestrial laser scanner and GIS data. In: 3D virtual reconstruction and visualization of complex architectures, Proceedings of ISPRS International Workshop 3D-ARCH 2009. Remondino F., 2011 – Heritage recording and 3D modeling with photogrammetry and 3D scanning. Remote Sensing, 3 (6): 1104-1138. <http://dx.doi.org/10.3390/rs3061104>
- Putha R., Quadrifoglio L., and Zechman E. (2012) Comparing Ant Colony Optimization and Genetic Algorithm Approaches for Solving Traffic Signal Coordination under Oversaturation Conditions, Computer-Aided Civil and Infrastructure Engineering, 27(1), pp. 14-28.
- Qu P.L. and Meyer G.W. (2008) Perceptually Guided Polygon Reduction, IEEE Transactions on Visualization and Computer Graphics, 14(5), pp. 1015-1029.

Reddy M., (2007). The graphics file format page. Available text online at:

<http://www.martinreddy.net/> Last accessed: 7th October 2015.

Reinhart, C. and Breton, P-F (2009) “Experimental validation of Autodesk® 3ds Max® Design 2009 and DAYSIM 3.0” *Leukos* 6 (1), 7-35.

Roncat A., Dublyansky Y., Spötl C., Dorninger P., (2011) Full-3D surveying of caves: a case study of Märchenhöhle (Austria). In: Marschallinger R. & Zobl F., 2011. Mathematical geosciences at the crossroads of theory and practice, Proceedings of the IAMG2011 Conference, September 5-9 2011, Salzburg, Austria: 1393-1403. <http://dx.doi.org/10.5242/iamg.2011.0074>.

Rosario B., Campomanes-Álvarez, O. and Damasa S. (2013) European Centre for Soft Computing, Gonzalo Gutiérrez Quirós s/n, 33600, Mieres, Asturias, Spain bDepartment of Computer Science and Artificial Intelligence, University of Granada, Periodista Daniel Saucedo Aranda s/n, 18014, Granada, Spain. Available online at:
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.413.2763&rep=rep1&type=pdf>. Accessed on the 19th October 2016.

Ross, M.T., Cruz, R., Hutchinson, C., Arnott, W.A., Woodruff, M.A., Powell, S.K. (2018) Aesthetic reconstruction of microtia: a review of current techniques and new 3D printing approaches. *Journal of Virtual and Physical Prototyping* Volume 13, 2018 Issue 2.

- Rossignac J. and Borrel P. (1993) Multi-resolution 3D approximations for rendering. In Modeling in Computer Graphics, pages 455–465. Springer-Verlag, June–July 1993.
- Rusu, R.,B. and Cousins, S. (2011) 3D is here: Point Cloud Library (PCL) IEEE International Conference on Robotics and Automation (ICRA)
- Ruy, B., Ramirez Rivera, A., Kim, J., Chae, O. (2017) Local Directional Ternary Pattern for Facial Expression Recognition IEEE Transactions on Image Processing. Volume: 26 Issue: 12
- Schodek D., Bechthold M., Griggs K., Kao K.M., Steinberg M. (2005) CAD/CAM Applications in Architecture and Design. John Wiley & Sons, Inc; New York: 2005. Digital design and manufacturing.
- Schroeder W. J., Zarge J. A., and Lorensen, W. E. (1992) Decimation of triangle meshes. In Computer Graphics: Proceedings SIGGRAPH '92, volume 26, No. 2, pages 65–70. ACM SIGGRAP.
- Sedano J., Berzosa A., Villar J.R., Corchado E.S., and De la Cal E. (2011) Optimising operational costs using Soft Computing techniques, Integrated Computer-Aided Engineering, 18(4), pp. 313-325.
- Seokbae S., Hyunpung P., Kwan H.L. (2002) Automated laser scanning system for reverse engineering and inspection. Int. J. Mach. Tools Manuf. 2002; 42:889–897.
- Sgambi L., Gkoumas K., and Bontempi F. (2012) Genetic Algorithms for the Dependability Assurance in the Design of a Long Span Suspension Bridge, Computer-Aided Civil and Infrastructure Engineering, 27(9), pp. 655-675.

- Shapiro, L. and Stockman, G. (2000) “Computer vision” , chapter 14: “3D models and matching.”
- Silva L., Bellon O.R.P., and Boyer K.L. (2005) Precision Range Image Registration using a Robust Surface Interpenetration Measure and Enhanced Genetic Algorithms, IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(5), p.762-776.
- Smeets, D., Claes, P., Vandermeulen, V. and Clement, J.G. (2010) “Objective 3D face recognition: Evolution, approaches and challenges”, Forensic Science International, Volume 201, Issues 1–3, pp. 125-132,
<https://doi.org/10.1016/j.forsciint.2010.03.023>.
<http://www.sciencedirect.com/science/article/pii/S0379073810001362>) Last accessed 23rd March 2018.
- Sokovic M., Kopac J. (2006) RE (reverse engineering) as necessary phase by rapid product development. J. Mater. Process. Technol. 2006;175:398–403.
- Sun Y., Paik J., Koschan A. and M. Abidi. (2003) Surface modeling using multi-view range and color images, Integrated Computer-Aided Engineering, 10(1), pp. 37-50.
- Talton, J., (2004) A Short Survey of Mesh Simplification Algorithms, pages 2-5
 Available online: <http://www.jerrytalton.net/research/t-ssmsa-04/paper.pdf> Last accessed 6th April 2018.
- Tan, K., Cheng, X. (2017) Specular reflection effects elimination in terrestrial laser scanning intensity data using phong model. Remote Sensing; 9(8):853.

- Tao H., Zain J.M., Ahmed M.M., Abdalla A.N., and Jing W. (2012) A Wavelet-Based Particle Swarm Optimization Algorithm for Digital Image Watermarking, *Integrated Computer-Aided Engineering*, 19(1), pp. 81-91.
- Thali, M., Dirnhofer, R., Vock, P. (2009) *The virtopsy approach: 3D optical and radiological scanning and reconstruction in forensic medicine*. CRC Press.
- Thomas D.M., Yalavarthy P.K., Karkala D., and Natarajan V. (2012) Mesh Simplification Based on Edge Collapsing Could Improve Computational Efficiency in Near Infrared Optical Tomographic Imaging, *IEEE Journal of Selected Topics in Quantum Electronics*, 18(4), pp. 1493-1501.
- Thompson, T.J.U, Norris P. (2018) A new method for the recovery and evidential comparison of footwear impressions using 3D structured light scanning. *Science and Justice*. Volume 58, Issue 3, May 2018, Pages 237-243.
- Tobler R., Maierhofer S., 2006 – A mesh data structure for rendering and subdivision. In: Jorge J., Skala V. (Eds.), 2006 - The 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG'2006) Short Communication Papers Proceedings: 157-162.
- Tognola G., Parazzinia M., Sveltob C., Ravazzania P., Grandoria F. (2003) A fast and reliable system for 3D surface acquisition and reconstruction. *Image Vision Comput.* 2003;21:295–305.
- Tong J, Zhou J, Liu L, Pan Z, Yan H. (2012) Scanning 3d full human bodies using kinects. *IEEE Trans Visual Comput Graphics*.18(4):643-650.

- Turk G. (1992) Re-tiling polygonal surfaces, Computer Graphics (SIGGRAPH '92 Proceedings), pp. 55-64
- Valstar, M., F., Sanchez-Lozano, E., Cohn, J. (2017) FERA 2017 - Addressing Head Pose in the Third Facial Expression Recognition and Analysis Challenge. 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)
- Varady T., Martin R., Cox J. (1997) Reverse engineering of Geometric models – an introduction. Comput. aided Des. 1997;29:255–268.
- Varshney, A. (1994) Hierarchical geometric approximations. Ph.D.Thesis TR-050-1994, Department of Computer Science, University of North Carolina, Chapel Hill, NC27599-3175.
- Watkins, A. (2011) “Creating Games with Unity and Maya: How to Develop Fun and Marketable 3D Games”. Focal Press.
- Watt, A. and Policarpo F. (2001) 3D Games: Volume 1: Real-Time Rendering and Software Technology Vol 1: Real-time Rendering and Software Technology.
- Weise T, Leibe B, Van Gool L. (2007) Fast 3d scanning with automatic motion compensation. Pp.1-8.
- Xiaodong T., Yuexian W., Xionghui Z., and Xueyu R. (2002) Mesh Simplification Based on Super-Face and Genetic Algorithm, Reverse Engineering. Advanced Manufacturing Technology, 29, pp. 303-312.

- Yeguas E., Joan-Arinyo R., and Luzón M. V. (2011) Modeling the Performance of Evolutionary Algorithms on the Root Identification Problem: A Case Study with PBIL and CHC Algorithms, *Evolutionary Computation*, 19(1), pp. 107-135.
- Zeng, N., Zhang, H, Song, B., Liu W., Li Y. (2018) Facial expression recognition via learning deep sparse autoencoders. Neurocomputing Volume 273, 17 January 2018, Pages 643-649
- Zhang M. (2007) Improving object detection performance with Genetic Programming, *International Journal on Artificial Intelligence Tools*, 16, pp. 849-873.
- Zhang Q. and Li H. (2007) MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition, *IEEE Transactions on Evolutionary Computation*, 11(6), pp. 712-731.
- Zhang, R. (2013) Edge collapse improved model-based three dimensional surface model data compression algorithm. Research Article at the Journal of Chemical and Pharmaceutical Research, 2014 - jocpr.com Available online at: <http://jocpr.com/vol6-iss7-2014/JCPR-2014-6-7-1206-1215.pdf> Last accessed on the 26 January 2015.
- Zhao, Z., Hwang, K., Villeta, J. (2012) - Proceedings of the 3rd workshop on Game cloud design with virtualized CPU/GPU servers and initial performance results. Available on: <http://dl.acm.org/citation.cfm?id=2287042> Last accessed on the 26 January 2015.
- Zhou H. and Liu Y. (2008) Accurate Integration of Multi-view Range Images Using K-Means Clustering, *Pattern Recognition*, 41(1), pp. 152-175.

Zitzler E. and Thiele L. (1999) Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach, IEEE Transactions on Evolutionary Computation, 3(4), pp. 257–271.

4D Dynamics (2012) Mephisto Extreme Quick Start Guide: 4D Dynamics (www.4ddynamics.com) Accessed on the 2nd August 2018.

Appendix I:

List of related research outputs

Constantinou, G., Wilson, G., Sadeghi-Esfahlani, S., Cirstea, M. “An Effective Approach to the Use of 3D Scanning Technology which Shortens the development time of 3D Models”. IEEE 2017 Joint International Conference on Optimization of Electrical and Electronic Equipment and the Aegean Conference on Electrical machines & Power Electronics (OPTIM-ACEMP). May 25-27 2017, Brasov, Romania, p1083-1088.

Constantinou, G. (2017) "3D Scanning and some applications" presentation. Faculty Seminar, Anglia Ruskin University, Cambridge. March 2017.

Constantinou, G. (2016) “3D scanning configuration and Optimisation” FST 5th Annual Research Conference presentation Anglia Ruskin University, Chelmsford. July 2016.

Constantinou, G. (2015) “OBJ file read/writer for 3D data processing” FST 4th Annual Research Conference poster presentation Anglia Ruskin University, Chelmsford. July 2015.

Constantinou, G. (2014) “Polygon Mesh Creation and Optimisation for 3D games” FST 3rd Annual Research and Scholarship conference. Short presentation Anglia Ruskin University, Cambridge. May 2014.

Appendix II

Source code:
OBJLoader.cpp
WavefrontLoader.h

OBJLoader.cpp

```

#include "WavefrontLoader.h"
#include <sstream>
#include <fstream>
#include <cassert>

WFOBJECT::WFOBJECT() {}
WFOBJECT::~~WFOBJECT() {}

// Wrapper Function- Load
int WFOBJECT::load(const char *filename)
{
    ifstream objFile(filename);
    for (string line; getline(objFile, line); ) {
        parseLine(line);
    }
    return 0;
}

void WFOBJECT::parseLine(string line)
{
    stringstream stringStream(line);
    // read in a line as a string , and assign the first string
    // to lineType which is also a string
    string lineType;
    stringStream >> lineType;

    if(lineType == "v")    // Vetrex
    {
        parseVertex(line);
    }
    else if(lineType == "vn") //Normal
    {
        parseNormal(line);
    }
    else if(lineType == "vt") // Texture
    {
        parseTexture(line);
    }
    else if(lineType == "f") // Face
    {
        parseFace(line);
    }
    else { // line that is neither a Vertex, nor Normal
        //nor Face.
        // do nothing
    }
}

void WFOBJECT::parseVertex(string line)
{
    stringstream stringStream(line);
    string lineType;

```

```

        stringstream >> lineType;

        assert(!lineType.compare("v"));

        Vector v;
        stringstream >> v.x;
        stringstream >> v.y;
        stringstream >> v.z;

        vertices.push_back(v);
    }

void WFObj::parseNormal(string line)
{
    stringstream stringStream(line);
    string lineType;

    stringstream >> lineType;

    assert(!lineType.compare("vn"));

    Vector vn;
    stringstream >> vn.x;
    stringstream >> vn.y;
    stringstream >> vn.z;

    normals.push_back(vn);
}

void WFObj::parseTexture(string line)
{
    stringstream stringStream(line);
    string lineType;

    stringstream >> lineType;

    assert(!lineType.compare("vt"));

    Vector vt;
    stringstream >> vt.x;
    stringstream >> vt.y;
    stringstream >> vt.z;

    textures.push_back(vt);
}

void WFObj::parseFace(string line)
{
    stringstream stringStream(line);
    string lineType;

    stringstream >> lineType;

    assert(!lineType.compare("f"));

    Face f;

```

```

for (string element; getline(stringStream, element, ' '); )
{
    if (element.empty() ||
        element.find_first_not_of("\r\n ") ==
            string::npos)
    {
        continue; // don't process whitespace tokens
    }
    // creating a FaceElement object and assigning
    // each member by parsing the '/' separator
    FaceElement faceElement;
    string item;
    stringstream ss(element);
    getline(ss, item, '/');
    faceElement.v = stoi(item);
    getline(ss, item, '/');
    faceElement.vt = stoi(item);
    getline(ss, item, '/');
    faceElement.vn = stoi(item);

    // Add this element to our Face elems
    f.elems.push_back(faceElement);
}

// Add the Face object to the faces member of the WFObjct
faces.push_back(f);
}

void WFObjct::write(ostream &outstream)
{
    outstream << "# Vertices:" << endl;
    for (auto &v : vertices)
    {
        outstream << "v " << v.x << " " << v.y << " " << v.z <<
            endl;
    }
    outstream << "# Normals:" << endl;
    for (auto &v : normals)
    {
        outstream << "vn " << v.x << " " << v.y << " " << v.z <<
            endl;
    }
    outstream << "# Textures:" << endl;
    for (auto &v : textures)
    {
        outstream << "vt " << v.x << " " << v.y << " " << v.z <<
            endl;
    }
    outstream << "# Faces:" << endl;
    for (auto &f : faces)
    {
        outstream << "f ";
        for (auto &e : f.elems) {
            outstream << e.v << "/" << e.vt << "/" <<
                e.vn << " ";
        }
    }
}

```

```

        ostream << endl;
    }
}

void WFObj::performOperations()
{
    // Do some stuff on vertices, normals, textures...
}

int main()
{
    WFObj obj;

    obj.load("purple sphere.obj");
    obj.performOperations();

    // Write to standard output (console):
    obj.write(cout);

    // Write the object into a file:
    std::ofstream ofstream("my_output.obj");
    obj.write(ofstream);
    return 0;
}

```

WavefrontLoader.h

```

#ifndef _WAVEFRONTLOADER_H_
#define _WAVEFRONTLOADER_H_

#include <iostream>
#include <fstream>
#include <vector>
#include <cstring>

using namespace std;

// #include <glut.h>

typedef struct
{
    float x;
    float y;
    float z;
} Vector;

typedef struct
{
    int v;
    int vt;
    int vn;
} FaceElement;

```

```

typedef struct
{
    vector<FaceElement> elems;
} Face;

class WFObject
{
    private: // Dynamic Variables to keep our object data in
        vector<Vector> vertices;
        vector<Vector> normals;
        vector<Vector> textures;
        vector<Face> faces;

        void parseLine(string line);
        void parseVertex(string line);
        void parseNormal(string line);
        void parseTexture(string line);
        void parseFace(string line);

    public:
        WFObject();
        ~WFObject();

        void performOperations();
        int load(const char *filename);
        void draw();
        void write(ostream &ostream);
};

#endif

```

Appendix III

Electronic files (on disk)