



Intelligent STL File Correction

by

GJ van Niekerk

DISSERTATION

submitted in fulfilment of the requirements for the degree of

MASTER IN NATURAL SCIENCE
OF
JOHANNESBURG
in

COMPUTER SCIENCE

in the

FACULTY OF SCIENCE

at the

RAND AFRIKAANS UNIVERSITY

SUPERVISOR: PROF EM EHLERS

October 2000

Acknowledgement

I would like hereby to convey my heartfelt gratitude to Professor Ehlers, for her patient guidance throughout this research study.

In addition, I would like to thank Mrs Van der Mast, for her help with the editing and proof-reading of the manuscript. My sincere thanks also to my parents and to Ansie, who supported me throughout this endeavour.

Finally, glory to God for enabling me successfully to conclude this dissertation.



Layered Manufacturing (LM), also known as “Rapid Prototyping”, is that process in terms of which a computer-designed model is created layer by layer with the aid of specific LM hardware. Telemanufacturing constitutes an extension of this technology that allows remote submission of manufacturing jobs or assignments across a communication medium, typically the Internet, to be built at the manufacturing bureau concerned.

The *de facto* standard of LM is the STL file. Simply put, this file consists of a number of triangles that are used to describe an object in its entirety. This file format has several advantages over other known formats and allows easy 2D rendering.

Unfortunately, however, the limitations of the latter format outweigh its advantages. Since the entire model is described in terms of a collection of triangles, the original geometry of the model is lost. As a result, a certain level of degradation will occur, especially around curvatures in the model. Although an increase in the number of triangles around such areas will enhance precision, it will also result in a much larger STL file.

Triangles that get lost somewhere inside the file could also give rise to holes, orphaned surfaces and zero-width walls in the projected object. It is vital, therefore, that the manufacturing bureau verify the correctness of the entire file before it is built in order to prevent machine time and materials from being wasted.

Instead of transmitting the entire file again, the bureau could attempt automatically to correct and repair less critical errors, thereby saving valuable resources and time.

“Layered Manufacturing” (LM), wat ook as “Rapid Prototyping” bekend staan, is daardie proses ingevolge waarvan ’n rekenaarontwerpte model lagie vir lagie met behulp van spesifieke LM-hardeware geskep word. Televervaardiging verteenwoordig ’n uitbreiding op hierdie tegnologie wat die langafstand-voorlegging van vervaardigingsopdragte deur ’n kommunikasiemedium, gewoonlik die Internet, moontlik maak, waarna die model by die betrokke vervaardigingsburo vervaardig kan word.

Die *de facto*-standaard vir LM is die STL-lêer. Eenvoudig gestel, bestaan dié lêer uit ’n aantal driehoeke wat gebruik word om ’n objek in sy geheel te beskryf. Hierdie lêerformaat hou verskeie voordele bo ander bekende formate in en vergemaklik die generering van 2D-beelde.

Ongelukkig weeg die voordele wat die formaat inhou, nie op teen die beperkings daarvan nie. Aangesien die ganse model met behulp van ’n versameling driehoeke beskryf word, gaan die model se oorspronklike geometrie verlore. Gevolglik sal ’n bepaalde vlak van degradering in die model plaasvind, veral wat krommings betref. Hoewel ’n vermeerdering in die aantal driehoeke rondom sodanige areas presisie sal verbeter, sal dit ook ’n veel groter STL-lêer tot gevolg hê.

Driehoeke wat êrens binne die lêer verlore raak, mag moontlik ook gate, losstaande oppervlaktes en mure sonder enige dikte in die geprojekteerde objek tot gevolg hê. Dit is daarom noodsaaklik dat die vervaardigingsburo die geldigheid van elke lêer sal verifieer alvorens die model vervaardig word ten einde te voorkom dat masjientyd en boumateriaal vermors word.

In plaas daarvan om die hele lêer weer te versend, sou die buro ’n poging kon aanwend om minder kritieke foute outomaties te korrigeer en te herstel, waardeur kosbare tyd en hulpbronne bespaar sou kon word.

Contents

1. Introduction to STL file correction

1.1	Introduction	1
1.2	Chapter organisation.....	2
1.3	Summary.....	5

2. The Layered Manufacturing (LM) process

2.1	Introduction	6
2.2	The new epoch: Layered Manufacturing (LM).....	10
2.3	Applications for LM.....	18
2.3.1	Molecular science	18
2.3.2	Engineering	19
2.3.3	Earth sciences.....	19
2.3.4	Mathematics and physics	20
2.3.5	Visualising complex occurrences	20
2.4	Problems and issues to be addressed.....	21
2.5	Summary.....	22

3. Making LM available to one and all: telemanufacturing

3.1	Introduction	23
3.2	Getting it there.....	23
3.2.1	In person.....	24
3.2.2	The postal service	24
3.2.3	Electronic mail (E-mail)	25
3.2.4	File Transfer Protocol (FTP).....	26
3.2.5	The World Wide Web (WWW).....	26
3.3	Summary.....	27

4. Transmission errors in telemanufacturing

4.1 Introduction28
4.2 Partial transmission29
4.3 Alteration of file contents30
4.4 Non-compliance with the agreed file format.....31
4.5 Incorrect protocol or incompatible systems32
4.6 Loss of file during transmission33
4.7 Summary.....34

5. Security in telemanufacturing

5.1 Introduction35
5.2 Safeguarding during transmission35
 5.2.1 User identification and authentication36
 5.2.2 Authorisation or logical access control.....37
 5.2.3 Confidentiality of information40
 5.2.4 Data integrity42
 5.2.5 Non-denial.....43
5.3 Summary.....44

6. The STL file format

6.1 Introduction46
6.2 The file structure.....46
6.3 Summary.....51

7. Compression in telemanufacturing

7.1 Introduction52
7.2 Compression for STL files53
 7.2.1 The Vertex Reuse Method54

7.2.2 The variable compression method 55
7.2.3 The redundancy compression method 58
7.3 Summary..... 59

8. An introduction to error checking and the prototype

8.1 Introduction 61
8.2 Principal aim of the project 62
8.3 Error checking 63
 8.3.1 Structural errors 63
 8.3.2 Geometrical errors 64
8.4 The interface 66
 8.4.1 Visualisation 67
 8.4.2 Error reporting 67
8.5 Complying with building constraints 68
 8.5.1 Coordinate range of the model..... 68
 8.5.2 Model size..... 68
 8.5.3 Model orientation..... 69
8.6 Summary..... 69

9. Structural-error checking

9.1 Introduction 70
9.2 Verifying the syntax of the STL file..... 70
9.3 Fixing syntax errors..... 73
9.4 Summary..... 74

10. Geometrical-error checking

10.1 Introduction 75
10.2 Removing duplicated triangles 75
10.3 Checking the structure against the vertex-to-vertex rule..... 76

10.4 Ensuring that each surface normal do indeed point outwards 83
10.5 Checking against Euler’s rule for legal solids 84
10.6 Summary 87

11. Checking the file against building constraints

11.1 Introduction 88
11.2 Ensuring that the coordinates of the solid fall within range 88
 11.2.1 The size of the model 89
 11.2.2 Distance from the table 92
 11.2.3 Part orientation with respect to coordinate ranges 92
11.3 Checking and possibly changing the object orientation 93
 11.3.1 Part precision 93
 11.3.2 Amount of building material used 94
 11.3.3 Time to completion 94
11.4 Ensuring that the file be compatible with the hardware 95
11.5 Summary 95

12. The prototype: STLComplete

12.1 Introduction 97
12.2 Syntax checking 98
 12.2.1 The scanner 98
 12.2.2 The parser 101
12.3 Geometrical checking 103
 12.3.1 Removing duplicated triangles 104
 12.3.2 Checking the object against the vertex-to-vertex rule 105
 12.3.3 Checking triangle orientation 108
12.4 The interface 109
12.5 Summary 114

13. Software comparison

13.1 Introduction	116
13.2 STLview version 7.0a.....	116
13.2.1 System requirements	116
13.2.2 Program features	117
13.3 Materialise Magics RP 4.3	118
13.3.1 System requirements.....	118
13.3.2 Program features	119
13.4 STeal version 1.2.....	120
13.4.1 System requirements.....	120
13.4.2 Program features	121
13.5 Summary.....	122

14. Conclusion

14.1 Introduction	124
14.2 Correcting STL files	124
14.3 Practicability of the STL format.....	125
14.4 Future research	126
14.5 Summary.....	127

Appendix A

Glossary

List of sources consulted

Figures

Figure 2.1:	Data flow during the engineering design process [2].....	6
Figure 2.2:	Evaluating a design on a CAD/CAM system [1].....	7
Figure 2.3:	Prototype of a side mirror tested in its final environment [5].....	9
Figure 2.4:	Effects of stair-stepping by increased layer thickness [7].....	11
Figure 2.5:	The stereolithography process.....	12
Figure 2.6:	The Fused Deposit Modeling (FDM) process.....	14
Figure 2.7:	The Laminated Object Manufacturing (LOM) process.....	15
Figure 2.8:	The Selective Laser Sintering (SLS) process.....	17
Figure 2.9:	The light-harvesting system, LH-II [9].....	18
Figure 2.10:	A functional gear prototype [9].....	19
Figure 2.11:	Earth as an LM model [9].....	20
Figure 2.12:	Interference between sinus waves [9].....	20
Figure 2.13:	The 2D representation of the fluid-flow model [9].....	21
Figure 2.14:	The 3D representation of the fluid-flow model [9].....	21
Figure 3.1:	The telemanufacturing process.....	23
Figure 5.1:	Identification and authentication on the Internet [20].....	37
Figure 6.1:	The wireframe representations [23].....	47
Figure 6.2:	Shaded representations [23].....	47
Figure 7.1:	A triangle mesh, demonstrating duplication in vertices [15].....	54
Figure 7.2:	Applying the 3D grid [28].....	56
Figure 7.3:	Choosing a value for q [28].....	56
Figure 7.4:	Hole-punching compression.....	58
Figure 8.1:	Framework of the prototype.....	62
Figure 9.1:	Layout of syntax-checking procedure.....	73
Figure 10.1:	Adjacent triangles in an STL file.....	76
Figure 10.2:	The vertex-to-vertex rule.....	77
Figure 10.3:	Approximated sphere with a missing triangle.....	80
Figure 10.4:	Triangle orientation of new triangle.....	81
Figure 10.5:	Badly damaged model.....	82

Figure 10.6: Correct and incorrect triangle orientation83

Figure 10.7: The prism..... 85

Figure 11.1: Robot leg after assembly [4]90

Figure 11.2: Orientation of a simple cube93

Figure 12.1: Detailed layout of the prototype.....97

Figure 12.2: Selecting an STL file to load110

Figure 12.3a: Viewing the model as a solid111

Figure 12.3b: Viewing the model as a wireframe.....111

Figure 12.4: A badly damaged model, with errors showing up in various colours 112

Figure 12.5a: Fixing the model.....113

Figure 12.5b: Revised model114

Figure 13.1: STLview interface (SDI option).....117

Figure 13.2: Materialise Magics RP 4.3 interface119

Figure 13.3: STeal by CIP software121



Tables

Table 6.1:	Technical assumptions obtaining to the binary STL representation.....	50
Table 6.2:	The binary STL representation	51
Table 7.1:	Compression with generalised triangle meshes.....	55
Table 12.1:	Examples of legal and illegal numbers in E-notation.....	100
Table 13.1:	Software comparison	123



Chapter 1

Introduction to STL file correction



1.1 Introduction

Homo sapiens is the only species on earth with the ability and drive on a large scale to model the environment to suit itself. We create sculptures, paint portraits and compose operas. We also apply our intelligence to more mundane matters in our constant endeavour to change, manipulate and improve our environment. The dawning of the Information Age, however, forever changed the way in which we work such changes. The computer, together with the Internet, also rendered obsolete many an old adage, such as “There is nothing new under the sun”, or “History repeats itself”, and heralded a new epoch characterised by an information explosion. The present study will be devoted to one aspect of the impact this explosion is having on humankind, namely to the manner in which information is presented to the end-user today.

Not so long ago, designers, engineers and architects making use of Computer Aided Design (CAD) technology to create a model had to content themselves with a flat, two-dimensional (2D) presentation of such model. This limitation gave rise to a myriad of errors, however, and not only hampered visualisation but also proved to be costly and time-consuming. Rapid Prototyping (RP) or Layered Manufacturing (LM) changed all that. Thanks to this relatively new technology, a model can now be fabricated before the designer’s very eyes, to the extent that the model can be held, felt and touched. The number of benefits to be derived from this technology is legion and it has already proven indispensable to numerous industrial applications [1].

Rapid Prototyping (RP) or Layered Manufacturing (LM) constitutes a process in terms of which a physical model is created thanks to the ability of a specific piece of equipment to receive input from a terminal on which the file describing the model in question is stored. Even though many different methods have been devised to implement this technology over the past few years, its underlying principles have remained unchanged [1].

Telemanufacturing constitutes an extension of this technology that allows remote submission of manufacturing jobs or assignments to a company or bureau with the required hardware. This technology, therefore, enables individuals and small companies to benefit from this process without having to acquire any of the expensive hardware involved [2].

The model is initially designed on a CAD system, whereupon it is transferred to an intermediate format, finally to be sliced and stored in a file format that the hardware can interpret. Although the intermediate step can be skipped, many bureaux on the Internet have adopted the StereoLithography (STL) file format as a primary medium for submitting job requests [1].

Unfortunately, such STL files usually are a haven for errors and anomalies, especially owing to their notoriously large size. Solving these problems is far from a trivial pursuit and special verification routines and geometrical-specific compression techniques have to be applied [2].

In the light of this, a section of the present study will be devoted to an introduction to the Layered Manufacturing and telemanufacturing technologies in general. In addition, the STL format and the problems by which it is plagued will be discussed in detail. Intelligent detection and correction methods will also be investigated and applied to a prototype software program.

1.2 Chapter organisation

The dissertation is divided into the following chapters:

a) Chapter 1: An introduction to STL file correction

This chapter allows an overview of the technologies to be discussed in the dissertation, as well as an overview of the manner in which the study is to be organised.

b) Chapter 2: The Layered Manufacturing (LM) process

The Layered Manufacturing (LM) process is discussed in detail in this chapter. A closer look is also taken at the various methods by which the said technology is implemented, as well as at the many fields in industry that have been benefitting from this technology.

c) Chapter 3: Making LM available to one and all: telemanufacturing

Telemanufacturing, that is, the technology that has made Layered Manufacturing available to a broader community, comes under discussion in this chapter. Here, the emphasis falls on the various methods used to transfer the file from the client to the bureau.

d) Chapter 4: Transmission errors in telemanufacturing

This chapter is devoted to the identification of the typical errors that occur during the transmission phase, as well as to the various ways in which to solve these problems.

e) Chapter 5: Security in telemanufacturing

Security breaches pose a considerable threat to many fields and have also cropped up in the telemanufacturing and LM arenas. This chapter is used to elaborate on such security issues and on ways in which to address them.

f) Chapter 6: The STL file format

One of the most widely used formats in the industry is that of the STL file. The syntax of the format is discussed in this chapter, as well as its pros and cons.

g) Chapter 7: Compression in telemanufacturing

The STL file format discussed in the foregoing chapter is notoriously large, with the result that good compression methods are required effectively to compress the file. This chapter is used to highlight the elements of the file that could be considered when designing or choosing a compression algorithm and to give a

few proven methods by means of which to achieve high compression ratios for the STL file format.

h) Chapter 8: An introduction to error checking and the prototype

This chapter is devoted to identifying the areas that are vulnerable to attack. It is also used to highlight the constraints on the STL format and to introduce the prototype developed for the present study, as well as the problem areas this prototype addresses.

i) Chapter 9: Structural-error checking

In this chapter, the STL file is defined in Backus-Naur form (BNF) and the concept of syntax checking is introduced.

j) Chapter 10: Geometrical-error checking

The vast majority of errors in STL files are still geometrical in nature. In this chapter, it is explained how each of the errors identified earlier occurs, as well as the possible solutions for them.

k) Chapter 11: Checking the file against building constraints

Layered Manufacturing has been implemented on many different types of machines and not all STL files can be processed by a specific device. This chapter introduces the areas that pose problems and changes that should be wrought to the model to ensure compatibility with a specific type of hardware.

l) Chapter 12: The prototype: STLComplete

This chapter is used to elaborate on the prototype and to discuss the algorithms used for detecting and fixing common errors in the STL file. The interface of the prototype is also discussed, together with the appropriate illustrations.

m) Chapter 13: Software comparison

This chapter is used to take a closer look at the software currently available in the market. In so doing, a set of parameters is compiled to compare software

with and the chapter is concluded with a comparison between software and STLComplete, the prototype that came under discussion in the previous chapter.

n) Chapter 14: Conclusion

The final chapter is used to formulate a conclusion to the author's findings and a hypothesis as to possible future developments in this technology.


1.3 Summary

“...prototypes not only enable products to be developed more quickly, but also result in products that are both higher quality and more effective in fulfilling their intended purpose in the marketplace.”

Maintaining the Lead in Manufacturing

Harvard Business Review

September-October 1994



Layered Manufacturing has changed the way in which prototypes are being manufactured. Although its associated hardware is still deemed expensive, telemanufacturing allows remote submission of files, which endows anyone with a credit card and a connection to the Internet with a powerful design tool. This growing technology has, in turn, necessitated the verification of submitted files in order to avoid the wastage of resources.

In the next chapter, the various LM technologies will be investigated, as well as those fields in the industry that have been benefitting from these technologies.

Chapter 2

The Layered Manufacturing (LM) process



2.1 Introduction

In the engineering field today, which includes disciplines such as mechanical, civil and electronic engineering, as well as various other forms of development and manufacturing, the construction of a part for some application comprises a step-by-step design method, commonly referred to as the “concept-design-production process” [2].

This process, graphically represented in figure 2.1 below, is illustrative of the iterative nature of the design process. Information that is constantly being fed back to previous steps is used to improve on and adapt each design [2].

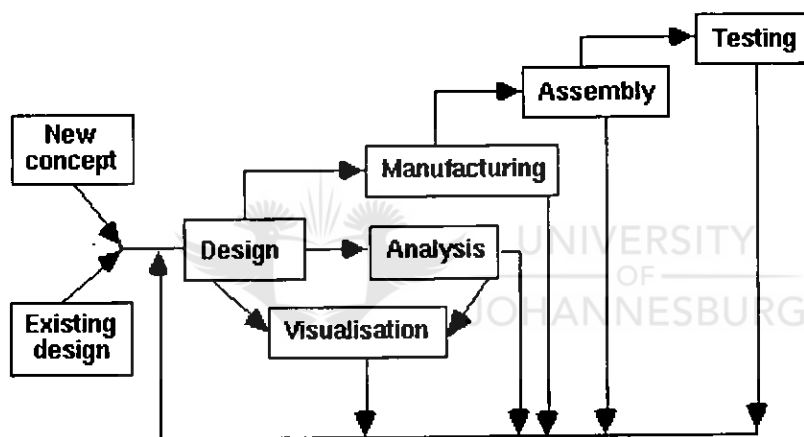


Figure 2.1: Data flow during the engineering design process [2]

The costs incurred during each step increase as one moves upward and to the right of the diagram. Iterating, though a complete manufacturing process to test for part correctness, is an expensive and time-consuming task. Unfortunately, the manufacturing of a physical prototype cannot be avoided entirely during the design process, at least not for the more complicated models. For this reason, methods need to be examined that will render the production of prototypes both economical and trustworthy. If the manufacturing of models cannot be avoided, improving upon the efficiency with which they can be created will improve upon the overall design process [2].

After a model has been designed, typically on a CAD system, a rough prototype needs to be constructed before final production of the model could commence. CAD systems today are exceptionally complex and enable developers quickly and effectively to design objects. Unfortunately, even the most complex and versatile CAD systems are still being dogged by the 2D limitation of computer monitors. We live in a 3D world, however, and it is for this reason that physical models play a pivotal part in the production cycle [2, 3].

To quote Mike Bailey, SDSC Senior Principal Scientist, UCSD Associate Professor and head of the TMF Project: "Just about every time people build a TMF model, they see something they missed in the computer graphics rendering. Computer graphics images and animations and even virtual-reality simulations (cannot) convey all the information that real objects do." From this, it is evident that physical prototypes play an important part in the design process [4].



Figure 2.2: Evaluating a design on a CAD/CAM system [1]

The physical model is especially useful for verification and visual-inspection purposes, which processes cannot always be executed in terms of the CAD application alone. Should an error be uncovered in the prototype, the designer

could correct it on the design and construct a new prototype, repeating the process.

A good prototype not only has to meet certain quality requirements, but it also has to do so under strict yet relevant conditions, which include the following:

a) Conditions obtaining to a specified margin of error

It stands to reason that a prototype with inaccurate dimensions will serve little or no purpose either for the verification or testing of the model. Although a certain margin of error is acceptable, that margin should be so small that thorough testing of the model would not be impeded in any way.

b) Conditions obtaining to the timeframe for prototype completion

In most cases of prototyping, it follows as a matter of course that the increased accuracy of a model would culminate in an extended production cycle of the prototype. If, however, the production time were unreasonably long, the costs incurred during the production phase would not be commensurate with the effort.

A strict time constraint on a project may compel the designer to accept the first operational model, if the manufacturing process would otherwise be dragged out for too long. Keeping the building cycle as short as possible will allow the designer to “experiment” until the optimal design be found.

c) Restrictions as to production costs

The total production costs of the prototype should be reasonable. These include both the material and labour involved in building the model. Again, an inordinately expensive model may result in a product that is far from optimal.

Manufacturing costs should not only be evaluated from a financial point of view, however. Cognisance should also be taken of other aspects such as environmental issues.

d) Requirements for testing purposes

All prototypes should be strong enough to allow rigorous testing of their models in the final environment. Unfortunately, a stronger prototype usually implies the use of more expensive material and more sophisticated hardware and, occasionally, even a protracted production time.

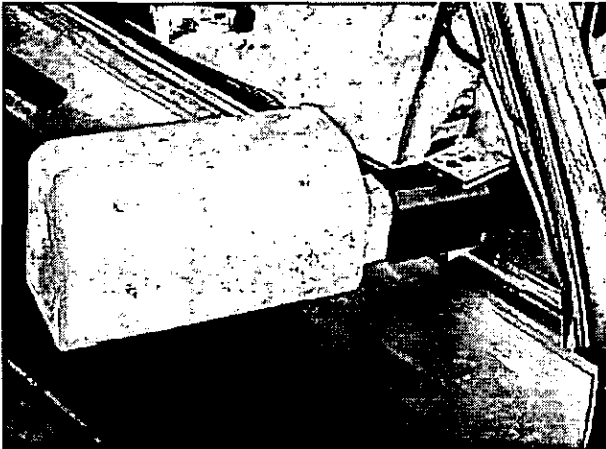


Figure 2.3: Prototype of a side mirror tested in its final environment [5]

Figure 2.3 above shows a prototype, in this case a side mirror of a vehicle, being tested in its final environment.

To date, various methods have been suggested and employed to construct these prototypes, each with its own pros and cons. Usually, while compensating for one constraint, another is neglected to some extent.

2.2 The new epoch: Layered Manufacturing (LM)

Layered Manufacturing has heralded an era in which new standards have been set for the production of prototypes in the manufacturing industry today. As was previously mentioned, most methods fail to meet all the requirements and constraints without neglecting one or more requirement to some degree. LM, however, strikes a balance between these requirements and allows economical and swift production of prototype after prototype. In addition, the LM process offers two unique benefits to designers: the ability to fabricate complex models that cannot be built by means of traditional techniques and a greater degree of automation during the design process [3].

The accuracy of a model depends on the machine used, but is acceptable for most applications. Usually, a higher accuracy rating can be attained in exchange for a longer manufacturing period, although even the longest production periods with LM are still very reasonable, compared to some of the production periods generated by means of the traditional methods.

Another aspect of LM that makes it an attractive alternative is the fact that it allows for rigorous testing before a model goes into final production. In terms of other, more delicate prototyping methods, testing is greatly hampered, if not impossible [1].

Although the LM process constitutes a marked improvement on many aspects of previous prototyping methods, it also poses new challenges to designers. One such challenge is the fact that the mere implementation of a geometrical algorithm is incapable of recognising the final purpose of the model. This makes it extremely difficult to tailor specific features in the decomposition without some form of human intervention. As these difficulties are slowly but surely being overcome thanks to continued research, the full potential of this technology will, however, be recognised and utilised during the manufacturing process [6].

In addition, although the hardware enabling the LM process is still relatively expensive (at the time of writing), once this asset has been obtained, the actual costs incurred in building a prototype will be much lower than when utilising conventional methods of part prototyping [1].

Although there are many different methods of LM, their basic fabrication process remains the same. The model is built layer by layer, starting from the bottom up. The width of each of the layers depends on the accuracy rating required for the prototype. (Varying layer widths have been suggested for and implemented in some applications.) The thicker the layers, the faster the model is produced, although very thick layering causes a stair-step effect that generally

detracts from the overall accuracy rating. The latter effect is graphically illustrated in figure 2.4.

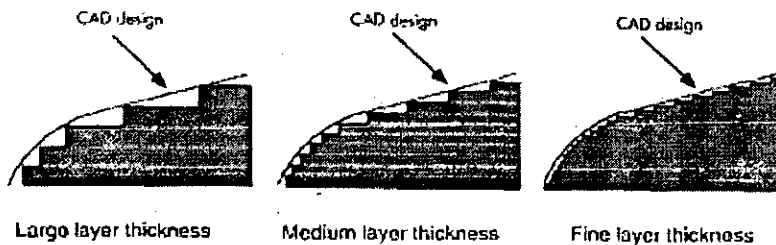


Figure 2.4: Effects of stair-stepping by increased layer thickness [7]

Before the model can be sliced and finally constructed, an STL file needs to be constructed from the original CAD design. An STL file can be defined as a collection of vectors describing a series of surface triangles in 3D space [8].

Although the simplicity of the STL file format makes it an attractive standard for LM, it suffers from a number of serious limitations. STL files are, for instance, notorious for having structural flaws. Holes in the object, zero-thickness walls and orphaned structures are commonplace. The accuracy rating of the model can, however, be enhanced by increasing the number of triangles. Resizing the model will, on the other hand, detract from its accuracy rating, since the object geometry is lost during the creation of the file. Despite these limitations, the STL format has become the *de facto* standard for LM that is supported and implemented by most systems [2].

Depending on the technology employed to build the actual model, supports may be required in the model and such supports are introduced in the model at this stage. The orientation of the model plays an important part in the supports required in it [1].

After the STL file has been generated and after supports have been added and checked for any anomalies, the object can be sliced in accordance with the thickness of its various layers. The said slice file consists of a collection of

closely spaced planes, each with a different z coordinate indicating its position within the object [1].

A final build file is then constructed, whereupon the process of constructing the object can commence. Following, a discussion on the actual layered-manufacturing methods, highlighting their respective pros and cons.

a) Stereolithography

In 1987, a new method of prototyping was devised. Also known as “3D printing”, stereolithography was the first commercially viable Layered Manufacturing technology, which is still widely used today [7].

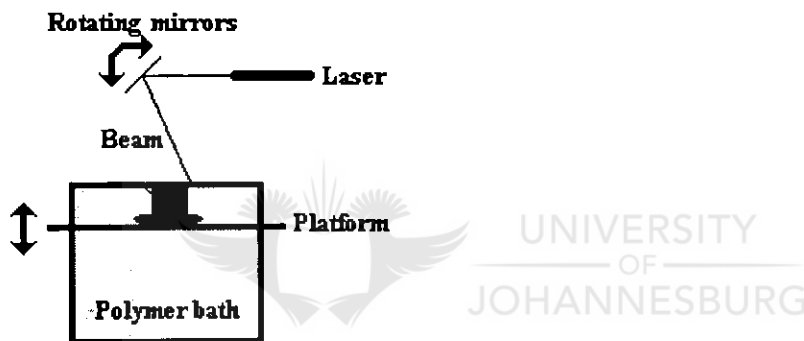


Figure 2.5: The stereolithography process

The building process is executed in a bath filled with liquid resin and in it a platform capable of moving in the z direction. Two types of lasers are generally used for stereolithography. Ultraviolet-laser technology is commonly used for the smaller applications, whilst a stronger helium-cadmium laser is used for larger, more sophisticated systems that require a higher accuracy rating [7].

Two mirrors, driven by a pair of galvanometer motors, are used to direct the laser spot downward, towards the surface of the liquid photopolymer. After having been sufficiently exposed, the liquid will be transformed into a solid state. On completion of each layer, the platform is lowered by a high-precision stepping motor. Liquid resin then flows over the recently solidified layer. A recoater blade or roller ensures that liquid resin to the thickness of exactly one

layer be added to the previous layer, and that the latter layer be evenly distributed across the entire surface. This new layer is then solidified by laser exposure again. This process is repeated until the entire model has been constructed from bottom to top [7].

The process of constructing a model in this fashion is known as “hatching”. A big drawback of stereolithography, however, is the fact that the models are subject to distortion in the form of shrinkage that occurs non-uniformly throughout the models. Fortunately, thanks to improved photopolymer compositions, computer software and laser technology, this part-distortion phenomenon has been greatly diminished, thereby improving the overall part accuracy rating [7, 8].

On completion of the model, the platform is elevated, revealing it in its finished form. After the excess liquid has been drained back into the vat, the supports that have been added to the design to facilitate the actual building of the model can be carefully removed. Care must, nonetheless, be taken not to damage the model whilst removing its support structures [7].

Finally, the model is placed in a Post-Cure Apparatus (PCA), where it is exposed to ultraviolet light of a certain wavelength. This step is taken to ensure that the model achieve optimal strength [7].

b) Fused Deposit Modeling (FDM)

Developed by Stratasys Inc., Fused Deposit Modeling (or “FDM”, for short) can be defined as that process in terms of which a thermoplastic material is heated to just above melting point. A nozzle is fed from a spool of thermoplastic material. Various types of material are available and can all be used on the same machine. Switching between materials can also be accomplished easily and quickly [8].

The melted substance is then extruded from the nozzle in the form of a thin ribbon. The melted plastic is deposited onto the previously built layer in this

form. The layer on which the new layer is constructed must, however, be maintained at a temperature just below that of solidification to ensure proper bonding. The nozzle is kept at a required distance by a computer-controlled platform that is moved in a downward direction as the layers are constructed [7].

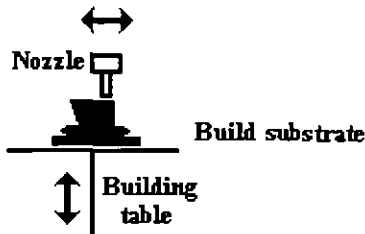


Figure 2.6: The Fused Deposit Modeling (FDM) process

The thickness of the layers is determined by the physical properties of the thermoplastic material used, the speed of the delivery head, the pressure with which the material is released and the diameter of the nozzle point from where the material is released. The overall precision of the model is greatly dependent on how closely the temperature of both the deposited material and the last layer is maintained. Model distortion in the form of rippling or, in the worst-case scenario, model collapse may occur if the temperature were to vary from the designated temperature. The nozzle should also never be allowed to become stationary above any given portion of the model. The high temperature at which the nozzle is operated will result in non-standard imperfections once the part has been completed [7].

FDM requires no high-powered laser technology, thus obviating the safety precautions associated with laser equipment. The material used for FDM, namely spools of plastic filament, has little or no environmental impact and does not require any handling or safety precautions. If the nozzle were sufficiently shielded, the elevated temperatures of both hardware and building material would hardly pose any risk. This makes it an ideal technology in scenarios where safety is at a premium [7].

c) Laminated Object Manufacturing (LOM)

This process was initially developed by Helisys Inc. As the name implies, Laminated Object Manufacturing (LOM) produces physical objects by the layering of thin sheet material. The material is typically supplied from a set of supply spools on either side of the evolving object. Each layer is joined to the previous layer by an adhesive that is both temperature and pressure sensitive. A roller, which is heated, ensures that the new layer adheres to the last by applying pressure from the top, thereby activating the adhesive [7].

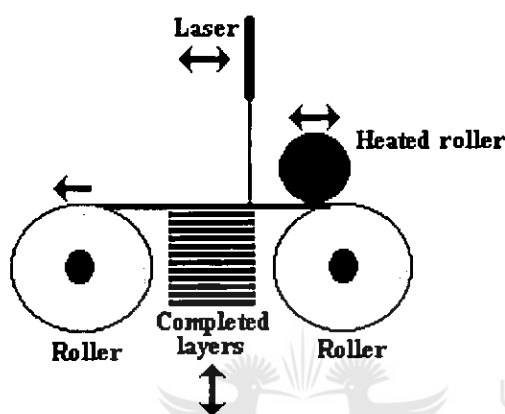


Figure 2.7: The Laminated Object Manufacturing (LOM) process

After the new layer has been applied, a carbon-dioxide laser traces the border, as specified by the build file. The laser beam renders each consecutive layer in this manner until the model is complete. The working platform is capable of moving in a vertical direction, which allows the laser to trace a layer at exactly the same height as that of the previous layer. The depth and width of the cut depend on the laser power, scan speed and physical properties of the laminated material used. The overall part accuracy rating is determined by the precision with which these parameters are controlled. If the finite width of the laser beam were ignored, the model would exhibit a small yet systematic flaw in each cross-section [7].

Because the entire model is submerged in a surrounding block of wasted material, no supports need to be constructed to keep the object from collapsing. Unfortunately, this has been known to waste a lot of material, which could

rather have been used to create a second or even a third model. Furthermore, to facilitate the removal of waste material, each consecutive layer needs to be diced, which takes up a lot of building time. Although the removal of waste material around large, solid parts generally poses no problems, special care should be taken to ensure that no part be damaged when material is removed near delicate and fragile sections of the model [7].

An attractive advantage of LOM is the fact that only the borders of each cross-section of the model need to be traced. This certainly saves a lot of time during the actual building process, which, in turn, justifies the dicing process that the prototype must undergo [7].

Although the laser can cut vertically through the sheet paper, cutting horizontally along two adjacent layers is not possible. Surfaces facing up or down will, therefore, still be connected to the adjacent layer, which may hamper the successful removal of waste material near that section. The integrity of the adjacent excess material could be compromised by cutting a closely spaced, crosshatch pattern that will facilitate the removal of such material [7].

A model produced by means of the LOM method will, in texture, smell and physical properties, resemble a model made of wood. Although LOM is a relatively inexpensive RP technology, special precautions must be taken to shield the invisible yet powerful and certainly dangerous laser beam used in LOM part building [7].

d) Selective Laser Sintering (SLS)

This process, which was developed at the University of Texas, produces RP parts in a similar fashion as stereolithography, but makes use of a fine powder, rather than a liquid resin. Using a counter-rotating roller, a thin layer of fine particles is spread evenly over the evolving object. A high-powered 50 watt carbon-dioxide laser is then used to fuse these particles together. In order to

minimise the laser output, the bed of powder is maintained at a constant temperature just below its fusing point.

The tray containing the incomplete object is constantly moved in a downward direction on a computer-controlled table to ensure that each layer be built at exactly the same height as that of the previous layer. In order to avoid oxygen contamination of the bonding surfaces, the building process is performed in an environmentally controlled chamber with an oxygen content of less than 2% [7].

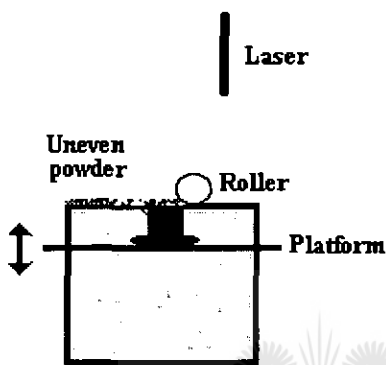


Figure 2.8: The Selective Laser Sintering (SLS) process

Unfortunately, the volumetric shrinkage for crystalline materials is quite considerable, which usually results in distortion of the part and imprecise dimensions of the finished model. The model is built in a raster fashion, resulting in a stair-step effect when constructing diagonal lines. A fine enough resolution will, however, modify this effect to an acceptable level. Furthermore, the discontinuous nature of the particles used will result in a rough finish, which may need finishing off once the part has been completed. Unfortunately, this will compromise the accuracy rating of the part to some extent [7].

The left-over unfused particles serve as a natural support structure for the part, even though anchor points may still be required for orphaned structures, which will be attached to the model in subsequent layers. On completion of the model, the left-over powder is removed with various brushes, cutting tools and low-pressure air. A wide range of materials can be used successfully as curing

material in SLS, which, in turn, makes for implementation in a much wider field than other RP technologies [7].

A wide range of materials (including metals) can be used during the SLS process, making it a choice method for future research [8].

2.3 Applications for LM

Following, a few examples of fields in the manufacturing industry that have been benefitting from this evolving technology.

2.3.1 Molecular science

LM can be applied at atomic level, thus allowing researchers to build complex molecular models. This is especially useful for visualisation purposes and for a thorough understanding of the nature of atoms and molecules.

Shown in figure 2.9 below is a 3D hardcopy of the light-harvesting system, produced by LM technology. The model was painted by hand on completion [9].



Figure 2.9: The light-harvesting system, LH-II [9]

2.3.2 Engineering

LM has made a huge impact on the engineering field, literally redefining the production process. Since prototyping of an incomplete model has now become much easier, verification has become more accurate and cost-effective.

Figure 2.10 presents just such an example. It depicts a gear system produced by means of LM equipment. The designer can now touch and move the various parts to get a better idea of the feasibility of the model [9].

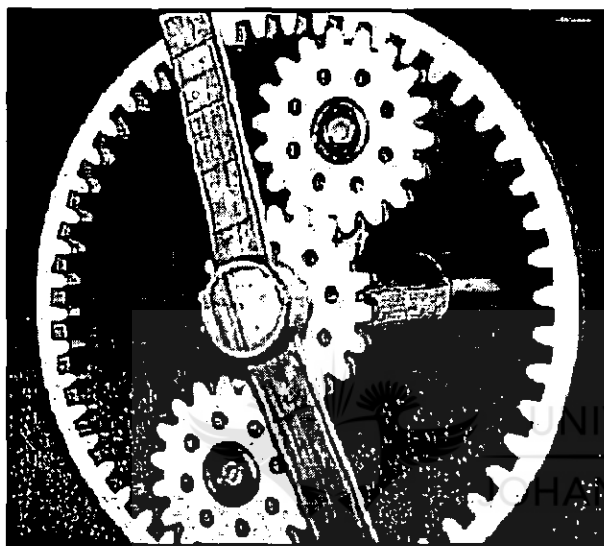


Figure 2.10: A functional gear prototype [9]

2.3.3 Earth sciences

When a satellite takes pictures of the earth, elevations at certain heights are depicted by various colours. These coloured images can, in turn, be converted by computer software into 3D objects and built by LM machines. Representing a region in this manner will allow geologists to conduct unique experiments and to visualise the geological dynamics with greater clarity [9].

Figure 2.11 shows a scaled-down elevation map of the earth. The African continent can easily be distinguished in the centre of the image.



Figure 2.11: Earth as an LM model [9]

2.3.4 Mathematics and physics

LM can be used to visualise natural occurrences and mathematical functions that have traditionally been restricted to paper. Figure 2.12 shows the interference that occurs when two sinus waves are generated at a certain distance from each other [9]:



Figure 2.12: Interference between sinus waves [9]

2.3.5 Visualising complex occurrences

Many events in nature are difficult or even impossible to visualise. A case in point is the flow of a liquid. In addition to quantities such as density, pressure and vorticity, which have only magnitude, a vector quantity such as velocity, which has both magnitude and direction, should also be represented. This model was traditionally visualised as a flat, 2D representation, with height indicating speed and colour indicating direction.



Figure 2.13: The 2D representation of the fluid-flow model [9]

Recently, at the American Institute of Aeronautics and Astronautics, a method was investigated for visualising the selfsame concept as a 3D model, thereby moving away from the 2D representation. Figures 2.13 and 2.14 depict what happened when the 2D image was represented as a 3D model. Adding colour to the model will enhance its representation even further [4].

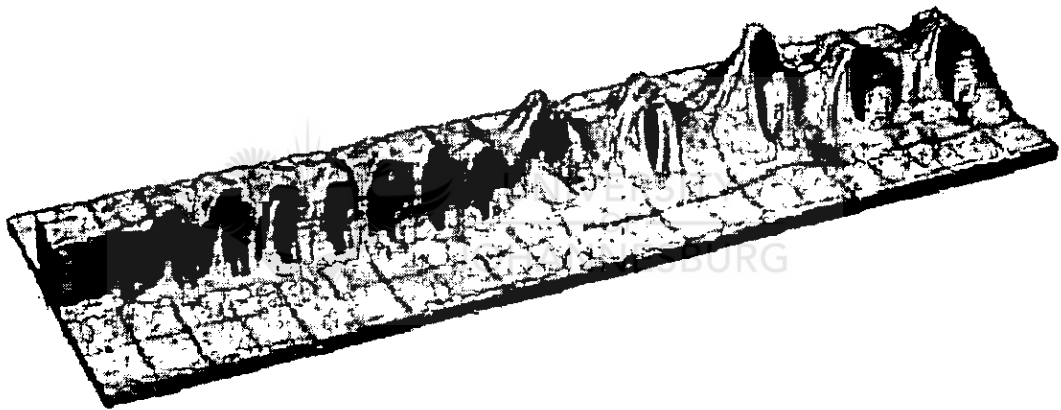


Figure 2.14: The 3D representation of the fluid-flow model [9]

2.4 Problems and issues to be addressed

Every new RP technology seems to rectify one or more shortcoming in its older version or versions, even though such new RP technologies oft-times create a brand-new set of problems, some of which are directly linked to the production of the new part, whilst still others can be traced back to the input file that was used during the manufacturing of the model. If a problematic input file were used that were fraught with errors, valuable resources would be lost, since the production would yield no more than an expensive piece of junk [1].

For this reason, the next few chapters will be devoted to a closer look at the aspects involved in checking for errors before embarking on the actual construction of the part. Emphasis will, for the purposes of the said investigation, fall on the verification and checking of the file.

2.5 Summary

Layered Manufacturing can be defined as that process in terms of which a model is created from a CAD model by a layering process. LM allows reasonably quick prototyping and constitutes a vast improvement on previous techniques. It allows enhanced automation of the entire design process and the construction of parts that could previously not be readily fabricated.

It also became evident from this chapter that a prototype is key to the design phase and that it has to be both economical and accurate. In the light of the wide variety of methods used in telemanufacturing and the unique features of each method, it is vital, too, to select the right hardware for the task at hand.

Following, a discussion on telemanufacturing as a means of making the LM technology available to a much wider audience.

Chapter 3

**Making LM available to one and all:
telemufacturing**



3.1 Introduction

Even though the actual production of prototypes by means of the LM technology may be much cheaper than by means of conventional methods, the hardware associated with the former technology still is expensive and out of reach for most. As with the very first computer systems, a qualified technician is needed to operate the LM system, which adds to maintenance costs even further [7]. For this reason, companies that can afford such machines and qualified personnel may decide to sell machine time in order to have their investment yield as big a profit as possible.

Telemanufacturing has paved the way for an even wider audience to benefit from the LM technology, and not only those fortunate enough to live in close proximity of an LM bureau [10]. Thanks to telemanufacturing, companies and individuals can now send a file that describes the part, together with their instructions, to a site where it can be constructed according to their specifications. Telemanufacturing is a growing technology that greatly aids the manufacturing process [11].

Following, a graphic representation of the telemanufacturing process:

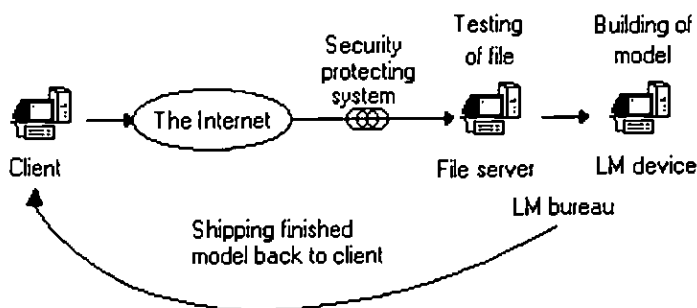


Figure 3.1: The telemanufacturing process

3.2 Getting it there

A wide range of technologies is available today by means of which to send a file to the manufacturing bureau. On receipt of such file, it must be checked for damage that might have occurred during the transmission process, especially as

some methods are more susceptible to errors than others. Following, an overview of the available technologies [10].

3.2.1 In person

The simplest, surest way of getting a file to the LM bureau is, of course, to deliver it in person. This entails saving the file on CD-ROM, a floppy disk or some other medium and taking it to the bureau. The instructions for building the prototype can be attached as a written request.

An advantage of taking the file to the bureau in person is that no complex verification methods are required to prove one's identity. This is, of course, a major concern when submitting the file by any other means.

A great drawback of this method, however, is distance. If a bureau were too far from one's physical location, taking the file in person would waste money and time. In addition, since there are so many different types of LM technologies in use, one has to ensure that the technology implemented to create the model will be suited to the task at hand. It is also very important to note that not all technologies will be able to create a model from a specific source. Some hardware requires supports or a specific orientation, for example, an STL file that has been prepared for an LOM device will not suffice for production on a stereolithography machine. This issue will be addressed in more detail later on in the dissertation [10].

3.2.2 The postal service

Despite all the advances in our hi-tech world, certain so-called "outdated" services are still very much in demand and, in some cases, even preferable to newer services. The postal service is one such service, as it can be used physically to transfer a file to a site. This method, however, is considered extremely slow in computer terms and is not always reliable. A local courier service may also be used, which would add to the reliability of the process and to the speed at which the file is delivered, albeit at a higher cost. In situations

where a connection to the Internet is unavailable, however, the postal (or courier) service may well be the only solution [10].

When sending the production file, exercising the above option may not be the best choice, but it would most likely be the only choice when the finished model has to be shipped back to the client (if not collected in person). A courier service will most probably be best suited to this task.

3.2.3 Electronic mail (E-mail)

Electronic mail has become an everyday service and, therefore, also a well-known and widely used medium for sending files to the LM bureau. The textual part of the E-mail message will contain the client's instructions, while the production file will be added to the mail message in the form of an attachment.

Unfortunately, E-mail messages pose a few problems, as their attachments may be damaged, especially if an attachment were rather large (which, in the case of STL files, they usually are). E-mail employs a message-switching protocol; that is, the message is stored at every node while the logic searches for the next node before the message is passed on. The message will, therefore, get lost or be severely damaged if some node were to store the message unsuccessfully owing to defective hardware or insufficient storage space [12].

Another problem with E-mail is that it is not practically possible to verify that the message has been successfully delivered. Although the receiver will often send acknowledgements, one could never be sure that it would be sent at all. Making sure that the E-mail message did indeed arrive successfully, one may be obliged to make a telephone call to the institution (thereby, in some cases, requiring an expensive international call).

Despite this, however, electronic mail still is a popular and widely used method for transferring files to LM bureaux. Thanks to the greater reliability of current Internet technology in general, problems involving E-mail have been steadily

overcome over the past couple of years, to the extent that it can now be regarded as a safe, convenient and affordable means of transferring files across very long distances [12].

3.2.4 File Transfer Protocol (FTP)

A marginally better means of transferring files over the Internet may be the well-known File Transfer Protocol (or “FTP”, for short), a protocol that was specifically developed for the transmission of files over the Internet (which makes it an attractive way reliably to send files).

Unfortunately, this is all it is: a file transfer protocol. No other latent functions have been included, which means that, after having submitted a job for production, the client has to inform the bureau by some other means of communication that a job has been submitted [10].

Although one could use a special program continuously to monitor the file server for newly submitted jobs, this is not always practical. Another solution may be to use the FTP in conjunction with E-mail, thus harnessing the best aspects of both technologies. The overhead costs incurred on both client and server sides may, however, not justify the effort.

3.2.5 The World Wide Web (WWW)

The World Wide Web constitutes a collection of uniquely identifiable file servers that hosts and provides documents (typically HTML) to the world. Thanks to the development of better Web browsers, CGI, Java and other technologies have also found their way into the information arena [12]. This added functionality has helped to make job submission over the WWW a reality for LM bureaux [10].


A great number of businesses are already deriving huge benefits from technologies such as these by allowing clients to send a job in its entirety via a Web browser, together with their instructions. Most technologies associated

with the WWW have since become *de facto* standards, thus facilitating compatibility between the various systems.

Error checking, encryption and visualisation can also be included, depending on the complexity of the Web site. Given these advantages, it is evident that (at least at the time of writing) the WWW constitutes the best and most preferred means by which to submit jobs over the Internet, and that it is the technology most widely used by vendors [10, 11].

3.3 Summary

Telemanufacturing is the key with which to unlock the wonders of LM technology to the whole world. Delivering the file to the vendor, however, is the most difficult task and careful consideration is required to ensure that the file be transmitted securely and that it would not be damaged or lost in transit. Some of the methods that could be used to deliver the job to the bureau include

- 
- delivering it in person
 - using the postal service
 - making use of E-mail messages
 - employing the File Transfer Protocol (FTP)
 - sending it from a Web page on the World Wide Web (WWW).

Important issues such as file integrity and security, as well as the speed and availability of the technology to be used for transmission, need to be considered before the file is submitted. Each of the technologies discussed in this chapter has its own strengths and weaknesses and the most appropriate one must, therefore, be chosen for each specific application.

The next chapter will be devoted to a discussion on the most common errors to manifest during transmission, as well as ways in which either to prevent or remedy them.

Chapter 4

Transmission errors in telemanufacturing



4.1 Introduction

Telemanufacturing has paved the way for smaller companies and individuals to benefit from the LM process. Unfortunately, however, the difficulties associated with the transmission of files over the Internet, as well as the security risks involved in doing so, have also become part and parcel of telemanufacturing. This chapter will, therefore, be devoted to a discussion on these errors and the problems they create.

When a file is transmitted from its source to its destination over the Internet, it passes through numerous servers, each possibly running a different operating system than that from which the file originated. The file is also sent through various media, each susceptible to different kinds of interference and specific weaknesses that might compromise the integrity of the information contained in the transmitted file [12].

For this reason, each file has to be verified for possible errors created in transit as soon as it arrives at the LM bureau. Another important factor that will receive our attention later is that of security. The number of people that witnesses the transmission request also poses a serious security risk. In today's competitive world, watertight security is vital [10].

Generally, the following types of errors could occur and must, therefore, be weeded out on reception of the transmitted file:

- The file was only partially transmitted.
- The file content was altered during the transmission.
- The file does not comply with the agreed file format.
- The file was transmitted via an incorrect protocol or an incompatible system.
- The file was lost in transit.

The Internet currently utilises the Transmission Control Protocol and Internet Protocol (TCP/IP) protocol suite to transmit information from source to destination. Even though these protocols do not form part of the Open System Interconnect (OSI) model, they do correspond to layers 4 and 3 of the said model respectively. These protocols, therefore, form part of a suite of protocols that is mainly used to effect communication between remote stations over the Internet. The Simple Mail Transfer Protocol (SMTP) defines the protocol used to effect correspondence via E-mail over the Internet, while the File Transfer Protocol (FTP) defines a protocol used for the transmission of files. Following, a discussion on the errors that could occur during data transmission whilst utilising these protocols [12].

4.2 Partial transmission

All information transmitted over the Internet is sent by way of packets. If the data set that needs to be sent were too large to fit into a single packet, it would be divided into multiple packets before transmission. When all the packets have successfully arrived at the recipient, they can be concatenated to reconstruct the original information. A file describing a part to be built on an LM machine is usually several megabytes in size and will, therefore, be divided into several packets to enable its transmission.

It may, however, still be possible for an LM system to process an incomplete file and to commence building the model that it describes. Should this be done, valuable resources would be lost, since the model would be incomplete and useless. Not much can be done to ensure that a file arrive at its destination in its entirety, but by careful examination thereof the building of a model from an incomplete input file can be avoided [10].

A simple method by which to correct this error is to add the actual size of the file at the beginning thereof. After the file has been reconstructed from the various packets, the actual size of the received file can be compared to the size stored in the header of the file. Should the two sizes fail to correspond, an error

has occurred and the file has not been transmitted in its entirety. Since this method merely checks for the partial transmission of files, it is seldom used.

In most cases, a Cyclic Redundancy Check (CRC) checksum is used for this task. CRC is based on polynomial division and is both reliable and effective. A CRC checksum is calculated and added in the header before the file is sent. At the destination, the CRC is recalculated and compared to the checksum stored in the header. If the two values were to correspond, the file was sent successfully. A CRC checksum will also validate the integrity of the data, with the result that, strictly speaking, CRC is an overkill if only the size of the file were of any concern. Thanks to its simplicity and high accuracy rating, however, it is a widely used safety mechanism [12].

Should it be detected, however, that a file was transmitted only partially, it should either be requested again in full or transmission should resume at the point where it was terminated. Sending the entire file again is often not necessary, especially if only a small percentage of the file has not been delivered.

Fortunately, many Internet servers now support “resuming on downloads”, which strategy could potentially save a lot of time if fully utilised. Software such as “Getright” by Headlight Software enables users to download a specific file in multiple sessions, supporting both HTTP and FTP. A future version of the software is expected to include an option in terms of which uploading a file can also be split into multiple sessions. This feature will, under certain conditions, serve dramatically to improve the submitting speed of files [13].

4.3 Alteration of file contents

During transmission, the contents of a file could, either accidentally or deliberately, be altered by malfunctioning hardware or by outside interference. A CRC checksum (as described above) is one of the best and most widely used

methods to check whether the contents of a file has been tampered with for any reason [12].

Fortunately, the integrity of data sent is effectively dealt with by the underlying TCP/IP. The TCP packet format contains a checksum in the header that is calculated and recalculated for every packet sent. Unfortunately, however, this only applies to individual packets and not to an entire file. Subsequently, the TCP/IP cannot be trusted to detect files that have only been partially transmitted. For this reason, a header containing at least a CRC checksum and/or the size of the entire file should be added in order to check the file, both for integrity and for full transmission [12].

4.4 Non-compliance with the agreed file format

Various file formats have been suggested for LM technology, many of which are in use currently. Although the STL file format (at the time of writing) is the format most widely used by vendors, it may well be replaced by an improved format, owing to its many disadvantages [14].

Before the file is finally submitted to the LM machine, it must be verified that the file has not only been correctly and completely received, but that it is in the correct format too. If the file were found not to be in the correct format, it must either be submitted again or converted into the correct format, on condition, naturally, that the bureau be equipped with the necessary software to do so. The said conversion must also be effected in such a way so as not to affect the size, accuracy rating or form of the model in any way. The following will help to define this idea more precisely:

Let F_1 be a function to convert a file from format **A** to format **B** and F_2 a function that will convert a file from format **B** to format **A**. If **X** were some file in format **A**, then:

$F_2(F_1(X)) = X$ must be true for all instances of X , for the file converter to be acceptable.

This is not always possible, however, since a specific format **A** may not necessarily contain all the information that a format **B** may contain. In such cases, information will inevitably be lost during conversion, whilst reverting to the original format may either be very difficult or impossible.

This conversion between various file formats and the problems associated with this exercise can, however, be avoided by agreeing with the bureau on a suitable file format before transmission. Errors may, unfortunately, still crop up if the user were to utilise error-prone conversion software, which constitutes yet another reason to verify the integrity of a file during the building process.

4.5 Incorrect protocol or incompatible systems

The protocols used by the sender and the recipient must either be the same or they must be compatible, failing which the file would not be transmitted correctly. If the Internet were used as underlying protocol, such contingencies would be catered for, but if a direct link were to be established between the source and the destination, the protocol must be agreed on and adhered to.

Another cause for file damage would be when a file is transmitted between two incompatible systems. An example of this type of error is when a binary STL file is sent from a UNIX-based machine that is set up for ASCII transmissions, instead of the BINARY format. The file would, as a result, be irreversibly scrambled [17].

STL files (especially the ASCII standard) are particularly large and cumbersome and compression techniques are often used to convert the files into smaller equivalents. Care must be taken, however, to use a standardised and compatible compression technique in order to enable the recipient to extract the original file. This is especially important if the operating systems between the

two systems were to differ. Compression techniques specifically developed for geometry-based files have been announced recently. These algorithms can potentially compress an STL file to a fraction of its original size [15, 16].

Once again, a CRC checksum is the most effective and preferred method by means of which to ensure that a file has been received successfully. Should the file be altered owing to an incompatible system or protocol during the transmission, the CRC would also be scrambled and the comparison would fail. Should this be the case, such file would have to be transmitted again [12].

4.6 Loss of file during transmission

Depending on the protocol used, the file can either be sent as a complete unit or it can be sub-divided into packets before being sent. On occasion, however, the file may not be delivered at all, with the result that means for verifying if the file had been delivered successfully or not need to be implemented.

If the file were only partially delivered or if fragments of it were lost (as in 4.2), the error(s) would be easy to uncover once the CRC checksum has been examined. The sender, however, needs to verify if the LM bureau had received the file, albeit only in part. The possibility does exist that the address of or the path to the bureau may be faulty and that, for this reason, the sender needs to determine if the file had been received.

The best way to achieve this is for the bureau to reply to the sender in the form of an E-mail message to confirm the delivery and successful extraction of the file and its accompanying instructions. Unfortunately, there is no way of knowing whether or not the bureau will reply, unless one makes a direct inquiry, which could be expensive and time-consuming, depending on the circumstances.

4.7 Summary

Transmission of files occurs over the Internet with dizzying frequency and since STL files are relatively large, they are susceptible to a whole range of errors. It is important, therefore, to ensure that the file be correctly transmitted to avoid the bureau from building a model fraught with errors.

Another important issue in telemanufacturing is that of security. It is not enough for a file merely to be transmitted error-free, it must also be sent in such a way that its contents remains intact and undisclosed to all but the interested parties. Security is also vital if payment were to be effected electronically. The next chapter will, therefore, be devoted to a detailed discussion on the issue of security.



Chapter 5

Security in telemanufacturing



5.1 Introduction

The question that springs to mind here is whether or not security in telemanufacturing really is an important issue. In today's world, where industrial espionage and sabotage are frequent and costly occurrences, it most certainly is. Competition amongst companies is exceptionally fierce and, should one party gain access to another's designs (even if they were unfinished), it would have devastating effects on the overall success of the resultant product. The latter party would gain an unfair advantage and would save much money, effort and time if it could gain access to the competition's designs. It is, therefore, imperative that communication between stations remain safe and secure and that effective security be maintained at LM bureaux too [10].

Another important factor is that of payment. LM bureaux need to be paid for services rendered and if the bureau and the client were great distances apart, this could become a thorny problem. In cases like these, it might be better to employ an electronic-payment method over the Internet. Thanks to continued research in the field of electronic commerce, this method can now be considered a feasible and efficient way of paying for services rendered.

5.2 Safeguarding during transmission

A reliable security system hinges upon five pillars that function individually in order collectively to make up the security system. If any of these pillars were to be flawed in any way, the system would have a security hole, regardless of how well the other pillars had been implemented. These pillars could, therefore, also be seen as a chain, in terms of which the trustworthiness of the entire security system would be equivalent to its weakest link. The five pillars of information security are as follows [18]:

- User identification and authentication.
- Authorisation or logical access control.
- Confidentiality of information.
- Data integrity.
- Non-denial.

As noted, these pillars also support one another in a logical sequence, a process to which the remainder of this chapter will be devoted.

5.2.1 User identification and authentication

A proper security system will always be able to determine exactly who is accessing the system at any given moment. This is vital for the following reasons [18]:

- To ensure that only certified users could access the system.
- Successfully to enforce logical access control.
- To enforce accountability or non-denial.

Depending on the manner in which the LM bureau renders its services, user identification and authentication will be affected to various degrees. Some bureaux operate in a closed environment, in terms of which only users that have been duly registered are allowed to access the system. In such cases, the system must be able correctly to identify each prospective user and to verify that such user is, in fact, who he/she claims to be [19].

A popular means by which to effect such identification and authentication is to request that the user type in his/her user name on the terminal (commonly a Web browser). The system will then check whether or not such user actually exist. If the user's name were not found on the database, login would be denied forthwith. If, however, the user's name were found, the system could authenticate if the user in question were indeed who he/she claimed to be. This is generally done with a password, which is known only to the legal party. Only if the password were to match the password on file would access be granted to the system, whereupon the user could proceed. The selfsame process is usually effected in one step over the Internet in a bid to minimise the packets sent between stations and to tighten security. The information thus sent between the systems must also be encrypted effectively to ensure that no third party would become privy to the user's login name and password.

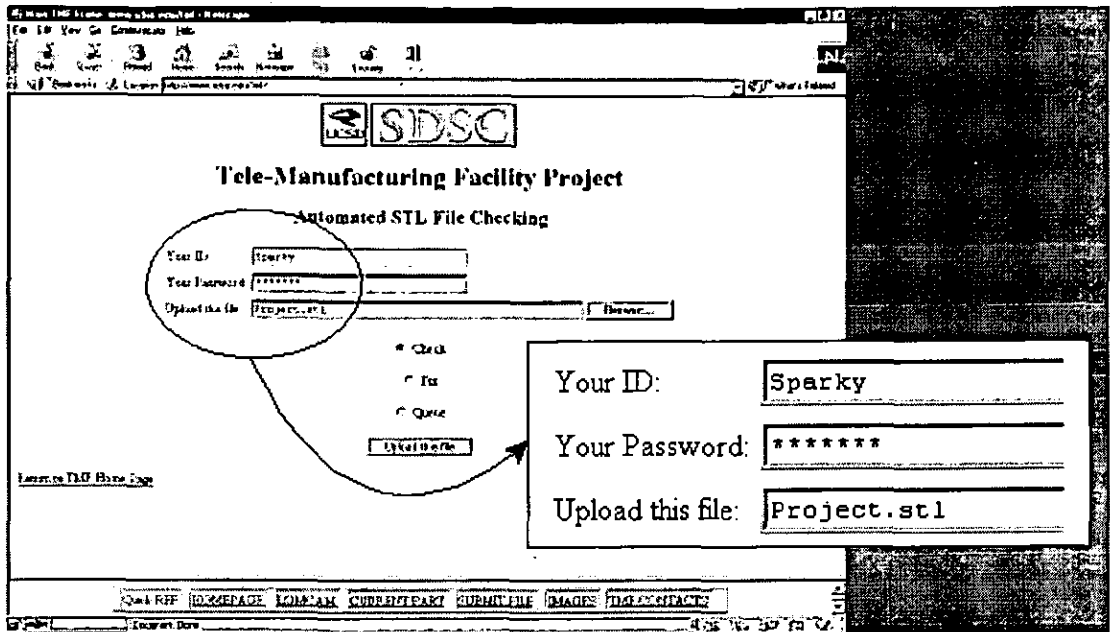


Figure 5.1: Identification and authentication on the Internet [20]

In a situation where a first-time user is requesting access to a system, as so often occurs in the realm of electronic commerce, the user will, naturally, not be registered. In order to account for such transaction (to furnish proof that the transaction did indeed take place), alternative steps must be taken.

5.2.2 Authorisation or logical access control

Access control constitutes a key aspect of the endeavour to ensure that only select users be allowed to perform certain functions on a system. After a user has been identified by the system, only those functions the particular user is allowed to perform must be made available to him/her. Access control is key to telemanufacturing for various reasons, including [18]

- the safeguarding of existing files in the queue from industrial espionage
- measures to prevent a system from building a model from a file that has been compromised in any way
- the ability to prevent users from erasing files (be it intentionally or unintentionally)
- the ability to screen technicians for the right to access the files.

Various models have been propounded and implemented to deal with the ever-greater security demands for access control and security, of which the following four models are most widely used:

a) Monitor Model

The Monitor Model constitutes one of the most basic authorisation models. The monitor is always online and validates each attempt at gaining access made between a subject (a user or computer program) and an object (an STL file or another program). The monitor, therefore, acts as a gateway between the subjects and the objects in the system.

The Monitor Model could, however, create a serious bottleneck in a system, since each and every attempt at gaining access has to be authorised by the monitor first. Despite this possible drawback, the Monitor Model is relatively easy to implement and maintain [18].

b) Information Flow Model

The Information Flow Model was propounded as an improvement on the Monitor Model, thanks to its ability to coordinate the flow of information between any two objects. The Information Flow Model acts as a filter that checks for both direct and indirect attempts at gaining access to information.

An example of such indirect attempt would be when a subject gains unauthorised access to a third object through legal access to a second object, on the assumption that the second object has access to the third. This is a trapdoor in the Monitor Model through which access could be gained to critical information [18].

c) Military Security Model

This model is based on the principle of least privilege, in terms of which a user is allowed access only to whatever information is required to perform a certain function.

The Military Model is based on a hierarchical system in terms of which every snippet of information is categorised according to its sensitivity and level of confidentiality. Four security levels are typically used, namely *Public*, *Confidential*, *Secret* and *Top Secret*. Each object is also catalogued into a specific compartment that assumes responsibility for and ownership of the information in question [18].

A subject is granted access to an object if and only if the

- security class of the subject is equivalent to or higher than that of the object
- compartment of the object is a subset of that which the subject has access to.

d) Bell and Lapadula Model

The Bell and Lapadula Model is the most popular model in the security industry today and shares many features with the Military Model, especially as both models make use of a hierarchical classification for subjects and objects.

Each subject and object is tagged or labelled according to its security clearance and sensitivity. As with the Military Model, multiple levels of security are defined and implemented, whilst the number of security levels varies, depending on the granularity required by the implementation [11].

Any environment can be described in terms of a set of objects and subjects. Suppose, for example, that Charles is the head engineer at an LM bureau, with Peter and Sally as his assistants. The bureau boasts an LOM machine and a stereolithography machine, as well as a file server that contains all the files (in some common format, such as STL) to be built by the latter machines. The sets of subjects and objects may comprise the following:

$S = (\text{Charles, Peter, Sally, Client}).$

$O = (\text{LOM machine, stereolithography machine, file server}).$

•

Every element of the subject set S must then be labelled to indicate the level of security to which the subject has access. Similarly, every element in the object set O must be labelled to indicate the level of security required for that specific object [18].

Access to an object is determined after having compared the tag of the subject with that of the object. Let $C(X)$ be a function that returns the security level or clearance of X , where X is a subject or an object. It can safely be assumed, therefore, that if $C(X) > C(Y)$, then the security level/clearance of X would be higher than that of Y (for example, X is *Secret*, whilst Y is merely *Confidential*).

A subject S_i has write access to object O_j iff $C(S_i) \leq C(O_j)$ (this is known as the “star property”) and read access to an object iff $C(S_i) \geq C(O_j)$. The latter prevents sensitive information from leaking through to lower levels when high-clearance subjects write critical information to lower-security level objects [18].

5.2.3 Confidentiality of information

Protecting the confidentiality of an object means that only authorised subjects would gain access to that object. This is especially important in telemanufacturing, where industrial espionage poses a very real threat. The confidentiality of both data in storage and of data in transit must, therefore, be maintained at all times [18].

Scrambling the original data in such a way that only the authorised party could extract it generally enforces the confidentiality of the data. The scrambled data is (albeit only in theory) completely worthless to any third party. The processes of scrambling and unscrambling data are referred to as “encryption” and “decryption” respectively [12].

Two types of encryption are generally used, namely symmetrical and asymmetrical encryption. The latter is more often used to enforce non-denial

and will be discussed later. Symmetrical encryption, on the other hand, is used to enforce data confidentiality and will be examined next.

- **Keyless symmetrical encryption:** a specific algorithm is shared between the parties and any secret information is encrypted using this algorithm. Usually, the same algorithm is used for encryption and decryption, but different algorithms (referred to as an “algorithm pair”) can also be used. The algorithm used is unique and must remain concealed in order to ensure confidentiality [18, 12].
- **Key-based symmetrical encryption:** this encryption method is similar to that of keyless encryption, except that a secret key accompanies the data and the algorithm. In order to decode the scrambled message, the party must be privy to the key, as well as to the decryption algorithm [18, 12].

Encrypted messages, however, still are vulnerable to attack. A third party could still intercept the message and, by choosing a random key, attempt to obtain the original data. Although exhaustive, this search method poses a very real threat to data security.

The longer the encryption key, the harder it would be to break the code. The rapid development in computer hardware over the past few years has also served greatly to complicate matters. Fortunately, the relationship between key size and key space is exponential and by merely increasing the key size by one character or digit, the key space is doubled.

Another problem associated with symmetrical encryption is the distribution of the secret key to authorised parties. Fortunately, asymmetrical encryption completely skirts this problem, which accounts for its being implemented more often [12].

5.2.4 Data integrity

Enforcing confidentiality is not enough to safeguard data in all circumstances. Although encryption will prevent unauthorised parties from gaining access to the real contents of the data, a determined hacker might still be able to alter or compromise the contents of the message. Data integrity will, therefore, guarantee that the submitted file is in its original form when the authorised party receives it [18].

It is vital in telemanufacturing to ensure that an undesirable party could not change the file in transit. Although the data remains confidential, valuable time might be lost if the contents of a file were to be changed. The LM hardware might even commence building the erroneous file, resulting in a useless prototype. It is crucial, therefore, to enforce data integrity and thus prevent industrial sabotage.

An authenticator scheme is usually applied to the encrypted message, which will prove that the message is authentic and that it has remained unchanged during the transmission process. The process is effected as follows:

The original message \mathbf{M} is first scrambled, using the encryption algorithm \mathbf{E}_K (with/without a secret key), which will yield an encrypted message $\mathbf{E}_K(\mathbf{M})$. The encrypted message is authenticated with \mathbf{C} , the authentication scheme. $\mathbf{C}(\mathbf{E}_K(\mathbf{M}))$ is obtained and is used as the authenticator for the message. $\mathbf{E}_K(\mathbf{M}) + \mathbf{C}(\mathbf{E}_K(\mathbf{M}))$ is now sent to the receiver through the network. The receiver has a copy of \mathbf{C} and can, therefore, compute $\mathbf{C}'(\mathbf{E}_K(\mathbf{M}))$ from the information sent. If $\mathbf{C}'(\mathbf{E}_K(\mathbf{M}))$ were to match $\mathbf{C}(\mathbf{E}_K(\mathbf{M}))$, the message could be deemed authentic [18].

Even though the said method is not foolproof, it often serves as the springboard for more advanced authentication schemes. It is also important in this respect to note that data integrity and confidentiality usually go hand in hand [18].

5.2.5 Non-denial

When a purchase is made or an order is sent, an analogue signature (a signature made by hand) is used to prove the identity of the respective parties. A signature is important, because it binds a party to a contract. Owing to electronic commerce having become a reality, other means had to be researched to enforce accountability for the parties involved. Asymmetrical encryption is used to construct digital signatures, with which a party can now sign an electronic document [12].

In case of asymmetrical encryption, a party is equipped with a unique key-pair. This key-pair consists of a public key, shown as $K_B(S)$, and a private key, denoted $K_P(S)$, for a specific subject S . These keys are mathematically related in such a way that any message encrypted with $K_B(S)$ could only be decrypted with $K_P(S)$, and vice versa.

The public key is constructed from the private key and the method used is such that going from the one key to the other is relatively easy, while going back from the constructed key is extremely difficult, if not impossible. The public key is made known through online directory services, whilst the private key is only known to S .

Digital signatures are now constructed as follows:

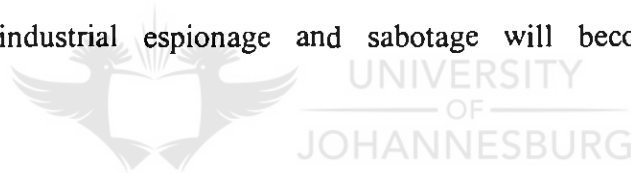
Suppose party A wants digitally to sign an electronic document and then send it off to party B , so that accountability can be enforced. Party A encrypts the message using his/her private key $K_P(A)$. This message is now sent to party B . It is important in this respect to note that the message is not confidential in any way, because the only key that can decrypt this message, namely $K_B(A)$, is publicly available. In order to ensure confidentiality and data integrity, the message needs to be encrypted a second time, as explained in the foregoing sections.

On having received the message, party **B** can prove its authenticity by decrypting it, using $K_B(A)$. If the original message were obtained, that message must have been sent by party **A**, if and only if the following two axioms hold:

1. If $E(M, K_1) = E(M, K_2)$ for all **M**, then $K_1 = K_2$, where $E(M, K)$ is an encryption algorithm taking a message **M** and key **K** as parameters (only one key could yield the encrypted message that could be decrypted by its counterpart).
2. K_P is the private key known only to party **A**.

5.3 Summary

Although information security has been the subject of extensive research over the past few years, it still remains a thorny problem in the realm of telemanufacturing. As more and more companies are making use of this new technology, industrial espionage and sabotage will become ever-greater problems.



An LM bureau with inadequate security is sure to lose clients and may even create a serious imbalance in the manufacturing industry if a fraudulent party were to gain an unfair advantage by breaching its security. Security is not only important during the transmission, but also at the bureau itself, so that only authorised personnel could gain access to sensitive data.

Another issue deserving of our attention is that of electronic payment. Clients must, naturally, pay for the services rendered by LM bureaux and this will probably be done digitally. It is important, therefore, that a client could do so safely and reliably.

Since the present study is aimed merely at touching upon the subject of security measures to be implemented during the telemanufacturing process, other sources should also be consulted for further information.

The STL file format is the *de facto* standard in the LM industry. Before shifting our attention to those means through which to remedy and correct errors in the file, however, we need to discuss the file format itself.



Chapter 6

The STL file format



6.1 Introduction

The STL file has become the *de facto* standard of both LM and telemanufacturing. Unfortunately, the format suffers from a number of serious defects that render it less than ideal for this task. Owing to these defects, the file is susceptible to a wide range of errors that will cause serious problems if negated.

Before error checking and correcting can be effected successfully, the structure of the file must be analysed and understood. The grammar or syntax of the format must be defined in order to process the file for verification and loading purposes.

6.2 The file structure

An STL file can be described as a collection of vertices representing a series of surface triangles in 3D space. (The terms *face*, *edge* and *vertex* are interchangeable with *triangle*, *vector* and *point* respectively.) Each triangle is also accompanied by a corresponding surface normal, which is needed to distinguish between inner and outer surfaces. This series of triangles is known as the “tessellated version of the prototype” and serves adequately to approximate the model [21].

The original CAD model is tessellated by an algorithm, such as the “adaptive subdivision” method, to yield an STL file. A higher accuracy rating can be achieved around curves and spherical surfaces by increasing the number of triangles in that specific region. Unfortunately, a higher accuracy rating also implies a bigger file. The file size, however, cannot be increased indefinitely, with the result that the storing and transmission of an STL file could pose a problem [22].

Since the object is defined in terms of a series of approximating triangles, the file does not constitute a perfect representation of the object. The format also does not allow for the storage of any geometrical data about the object.

Although the former error could be corrected to some degree, the lack of geometrical data in the format is the source of numerous problems that are difficult properly to address. The loss of the geometrical properties of the model clearly is one of the biggest disadvantages of the STL file format [2].

The following example is illustrative of the effect when a sphere is tessellated by a limited number of faces, which, in fact, requires an infinite number of triangles before it could be represented perfectly. In the example, the image on the extreme left displays the sphere when it is approximated by six faces, following which the accuracy rating of each subsequent image is increased by a factor of four. The image on the extreme right, therefore, contains no fewer than 1 536 faces [23].



Figure 6.1: The wireframe representations [23]

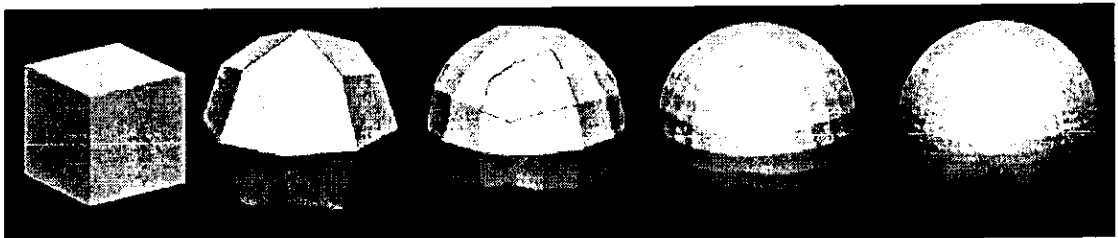


Figure 6.2: Shaded representations [23]

The simplicity of the file, however, allows the ASCII variant thereof (shortly to be discussed) to be transported between systems of various architectures and operating systems. In addition, the format does not require any calculation overhead, which will save system time, especially on larger files. The wide use of the format also makes it a good choice when submitting a file to a telemanufacturing bureau [14].

Two variants of the STL file are widely used. The information contained on the object is identical for both file types [8].

a) ASCII format

This format usually engenders rather large files. Thanks to the nature of the file, however, it is easy to read and can be edited manually. The ASCII file also is more suitable to the Internet environment, since ASCII usually is much more platform-independent than its binary counterpart [8].

Unfortunately, the file does not contain a trustworthy header to identify the file and complete parsing is required successfully to identify the file.

b) Binary format

Although the binary format is much smaller in size, it is platform-dependent and can, therefore, only be successfully used between systems of similar architectures. The binary file is not as susceptible to syntactical errors as its ASCII counterpart, however, with the result that it is generally not necessary to parse the file.

STL files in the ASCII and binary formats can, for transmission and archiving purposes, be compressed very effectively into a much smaller equivalent. (This is especially true for the ASCII variant.) Specific geometry-based compression techniques have been developed and implemented successfully [24].

Apart from that, the repetitive nature of specific strings in an ASCII STL file makes Lempel-Ziv or any other substitution-based technique an attractive method. Huffman encoding will also yield very good results, since certain characters have a far higher incidence than others. Characters that occur in the file more frequently will be stored using a shorter bit string than those with a lower incidence. This method will result in a much smaller file than when each character is stored in exactly the same number of bits. Compression techniques will be discussed in more detail in the next chapter [12].

It is important to take into consideration the platform on which the file will be decompressed before applying any form of compression to it. If a file were compressed on a specific platform, it might be rendered incompatible with other systems. Fortunately, there are a number of compression methods in terms of which the file will remain platform-independent, which methods could be employed as and when required.

The ASCII STL file is underpinned by the following format [2]:

solid [name of solid]

facet normal N_1

outer loop

vertex V_1

vertex V_2

vertex V_3

endloop

endfacet

facet normal N_2

outer loop

vertex V_4

vertex V_5

vertex V_6

endloop

endfacet

 *

 *

facet normal N_i

outer loop

vertex $V_{(i+3-2)}$

vertex $V_{(i+3-1)}$

vertex $V_{(i+3)}$

endloop

endfacet

endsolid



The file must begin and end with the reserved words *solid* and *endsolid* respectively. The reserved word *solid* is occasionally followed by a descriptive name of the specific model that the file represents, although such identifier is not compulsory in the STL standard.

Following, the entire model is defined between these delimiters as a series of triangles, which, in turn, consists of a number of vertices. All reserved words in the STL file are in lower case [21].

The vertices V_n represent the three corners of a surface triangle i describing that section of the model. The x, y and z components of each of these vertices are in Cartesian coordinates and should all be positive floating-point numbers. The normal vector N_i is a unit vector of length 1 based at the origin and accompanies each defined triangle. This normal is determined by calculating the vector cross product between any two vectors within the triangle [21].

Unfortunately, the STL format is not very robust in the sense that a file will still pass as a legal STL file even when the object has been severely compromised [25].

The binary STL file is underpinned by the following format:

Table 6.1: Technical assumptions obtaining to the binary STL representation

Unit type	Size	Lower boundary	Upper boundary
Bit	On/Off state	0	1
Byte	8 bits	0	255
Unsigned integer	2 bytes	0	65,535
Unsigned long integer	4 bytes	0	4,294,967,296
Float	4 bytes IEEE	$\pm 1.5 * 10^{-45}$ (7-8 significant digits)	$\pm 3.4 * 10^{38}$ (7-8 significant digits)

Table 6.2: The binary STL representation

Description	Size
File header	80 bytes
Number of triangles	Unsigned long integer
<i>For every triangle in the model</i>	
Normal	3 floats
First vertex	3 floats
Second vertex	3 floats
Third vertex	3 floats
Attribute	Unsigned integer

The format of the STL file suggests a highly simplified object-orientated approach, in the sense that each *solid* or real-world object consists of a number of objects called *facets* (or *triangles*), with each facet consisting of exactly three *vertices*. The three vectors defining the triangle are now constructed between these points.

6.3 Summary

The STL file is the *de facto* standard of both LM and telemanufacturing. Even though it suffers from several weak points, the simplicity of the file has made it an attractive format with the LM and telemanufacturing technologies. There are two specifications of the STL file format: a text-based or ASCII file that is more compatible between systems but larger in size, and a binary representation that can only be used between compatible systems but which is much smaller in size.

Having discussed the format of the STL file, the next chapter will be devoted to an introduction to several compression methods that can be applied to a file in order to minimise its overall transmission time and the space required for its storage.

Chapter 7

Compression in telemanufacturing



7.1 Introduction

One of the greatest drawbacks of the STL file format is the large-sized files it tends to generate. Since any curvature in an object can be represented by an infinite number of triangles, the number of triangles necessary to achieve an acceptable accuracy rating oft-times causes an object file of average complexity to become too cumbersome [16].

These large files, in turn, tend to complicate Internet transmission. In addition, larger files require more storage space and if a telemanufacturing bureau were to receive a high volume of job requests all at once, a serious problem could be created.

Fortunately, however, compression techniques applied to STL files can effectuate substantial savings in the storage space required. Two categories of compression techniques are available to compact files, namely that of general compression techniques and that of format-specific compression techniques.

a) General compression techniques

The most widely used compression agents, such as “WinZip” (a Windows-based utility), “Gzip” and “Arj” (a multiplatform utility), allow for the ready compression of any type of file. The compression effected by means of this category of compression techniques is loss-less, which means that no loss of information will occur during the compression and decompressing processes. Since these utilities are readily available, many telemanufacturing bureaux require that all files must first be compressed by means of a specific compression agent before their submission [26, 27].

The general nature of these algorithms, however, limits the compression ratio to be attained and more specific algorithms should be applied better to compress STL files. Another problem associated with general compression techniques is that they are usually limited to system architecture. In other words, a file that

has been compressed by means of a specific architecture could only be decompressed by means of a compatible system [24].

b) Format-specific compression techniques

A specific compression algorithm is an algorithm that can be applied to a specific file format only. Although some of these algorithms cause a loss of some information, such losses are generally acceptable. Extremely high compression ratios are commonplace in terms of this category of compression techniques. The JPG compression for high-dimension images and MP3 compression for audio are well-known examples of such compression schemes attaining high compression ratios (both of which lose some data in return for such high compression ratios).

Fortunately, the geometry hidden in the mesh of triangles in an STL file allows for specific compression algorithms to be applied to the file, resulting in file sizes that are mere fractions of the original sizes [15].

In case of ASCII STL files, the compression can also be effectuated in such a way that the files remain platform-independent even after having been compressed. Although the compression ratios will not be as high, the files will still be significantly smaller.

7.2 Compression for STL files

This section will be devoted to a discussion on those compression methods specifically propounded for the STL file format, many of which can be combined to achieve excellent compression ratios. After having applied geometrical compression, a general compression method may also be applied even further to decrease the size of a file.

7.2.1 The Vertex Reuse Method

A generalised triangle mesh is a compact representation of 3D geometry. It is based on the premise that many vertices will be used several times by different edges [15].

In figure 7.1 below, a triangle mesh is shown with numbered vertices. A typical STL file would list each triangle consisting of three vertices. Each vertex is a point in space and consists of an x, y and z component. As was mentioned in chapter 6, each component takes up 4 bytes of storage (binary format). This implies that each vertex takes up a total of 12 bytes. Since a vertex is shared by a number of triangles, much space is wasted owing to duplication.

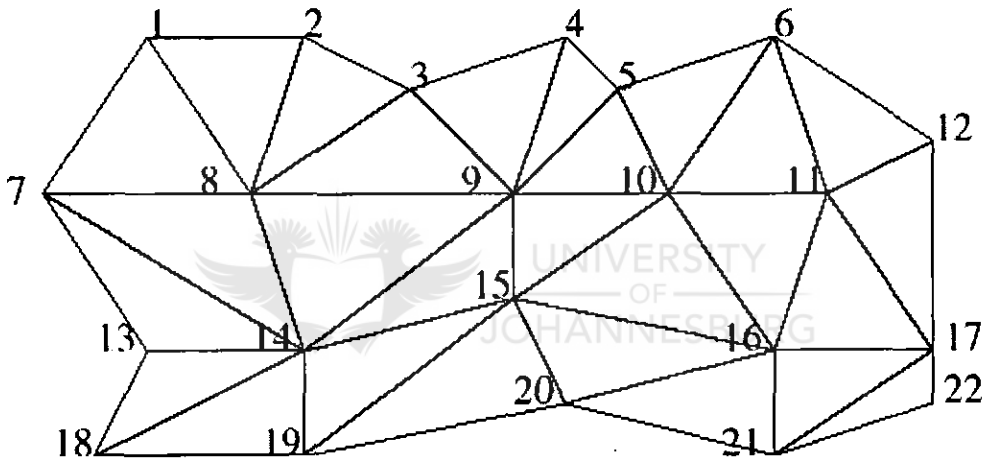


Figure 7.1: A triangle mesh, demonstrating duplication in vertices [15]

An STL file can easily be compressed by using a “mesh buffer” on which to store those vertices with the highest incidences. The “mesh buffer” is a lookup table with a predefined size. A pointer value is used instead of having to list the entire vertex for each triangle. In this way, a table with 256 entries requires an 8-bit, or a 1-byte pointer [15].

Although larger buffers could store more vertices, they require pointers that take up more space. Smaller buffers, on the other hand, require smaller pointers, but cannot hold as many vertices. A good balance should, therefore, be struck between the size of the buffer and the size of the STL file.

Employing this method effects a substantial saving in the space taken up by an STL file. Table 7.1 is illustrative of the saving potentially to be effected by this method. In this example, each vertex is assumed to be shared by six triangles. In addition, the lookup table is assumed to be fully utilised.

Table 7.1: Compression with generalised triangle meshes

Size of lookup table	Size of pointer	Saving
16 entries	1 nibble (4 bits)	912 bytes
256 entries	1 byte	13,824 bytes
65,536 entries	2 bytes	3,145,728 bytes
16,777,216 entries	3 bytes	704,643,072 bytes

Using a mesh buffer constitutes an excellent method by means of which to compress STL files, and one stands to derive the following benefits:

- Completely loss-less compression is achieved.
- ASCII STL files still are platform-independent.
- Compression and decompression require little computational overhead.
- Can be used in conjunction with other general and geometry-based compression methods, thereby achieving high compression ratios.

7.2.2 The variable compression method

Another method by means of which to compress STL files is by variable compression. This method can be used in conjunction with the Vertex Reuse Method described in the previous section. The variable compression method is used to compress a file by reducing the space that the numbers take up, albeit with a small loss in its accuracy rating. Since the entire object consists of nothing but a series of structured vertices, high compression ratios could potentially be achieved.

Fortunately, some applications allow minimal degradation of the object and this method can be employed in such cases. Research in this domain has shown

that to reduce the number precision rating from 32 to 16 bits usually results in little loss of quality [15].

The first step in the compression process is to normalise the entire object, so that all its coordinates fall within the range $[-0.5, 0.5]$. On decompression, the object can be translated and resized to meet the building constraints in question [15].

The next step in the said process involves a reduction in the space taken up by each vertex by truncating the least significant m bits of the position components, where $m = 32 - q$ (with $q \leq 32$, a predefined integer). This can be viewed as overlaying a 3D grid over the object, as shown in figure 7.2 [15]:

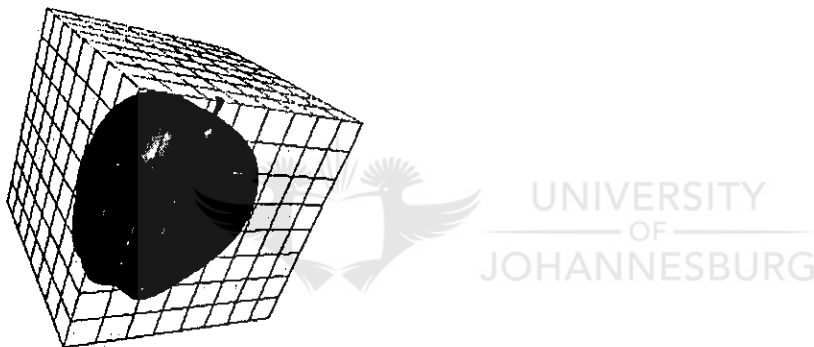


Figure 7.2: Applying the 3D grid [28]

The value of q must, however, be carefully chosen. Selecting a q value that is too small will result in a badly deformed model, while too large a value will result in a non-optimal compression ratio [15] (see figure 7.3):

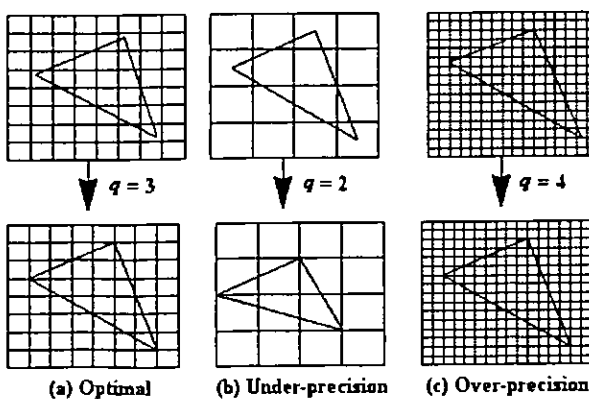


Figure 7.3: Choosing a value for q [28]

This method has been modified, however, and many aspects of the algorithm have been improved, including:

a) Selection of the q value

The manual selection of the q value used to be a trial-and-error process, which constituted a serious drawback. One way is manually to select an error threshold and then to request that the system select a value for q , such that all fluctuation in coordinate adjustments be within this threshold. In this way, the file can be compressed optimally [15].

b) Region-based compression

Not all regions of a specific part have the same number of triangles. More triangles are present around curvatures and round corners. An improvement on the algorithm is separately to compress these areas, with a different value for q . This will result in an optimally compressed file [15].

The variable compression method allows for good compression, depending on the overall model geometry. Unfortunately, a certain degree of precision is lost and a fair amount of computational power is required during the compression phase. One stands, nonetheless, to derive the following benefits from employing this method:

- Region-based compression.
- The q value allows various levels of compression.
- The compression level is chosen automatically after having specified the error threshold.
- Can be used in conjunction with other general and geometry-based compression methods to achieve high compression ratios.
- No decompression is necessary.

7.2.3 The redundancy compression method

An uncompressed STL file contains huge chunks of redundant information. These serve, naturally, to inflate the STL file [29]. By removing all the redundant information, a marked reduction can be effected in the file size.

a) Removal of normals

Each triangle in the STL definition has a surface normal that distinguishes the outside of the object from its inside. Each of these normals takes up 12 bytes of space. Since each normal can easily be calculated by taking the cross product of any two vectors of that specific triangle (assuming the triangle orientation is correct), as much as 20% of space can be saved (depending on the type of STL file) by omitting the normals in the file once the triangle orientation has been verified.

b) Hole-punching

This method is, in many respects, similar to that of re-using vertices. Each edge is shared twice between two triangles. The size of the STL file can, therefore, be greatly reduced by removing one of the triangles sharing each edge. As long as the set of remaining triangles represents each edge contained in the entire model, there will be ample information to reconstruct the file in its entirety. This concept is illustrated in figure 7.4:

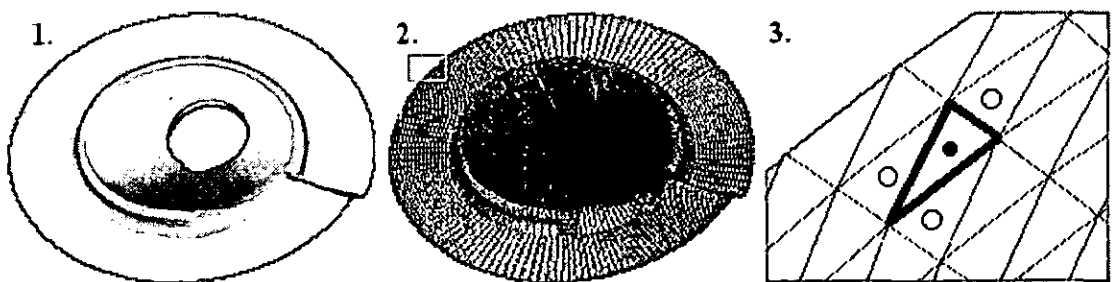


Figure 7.4: Hole-punching compression

A high-precision design is depicted in the extreme left image of figure 7.4. Once the entire triangle mesh is shown, the complexity of the model becomes evident, as depicted in the second image in the same figure.

In the third image, an enlarged section from the second image is shown. The individual triangles are now distinctly visible. On this image, each of the edges in the triangle containing the black dot is shared with three other triangles, each marked with a white dot. Since each of these neighbouring triangles contains the same edge, the triangle marked with the black dot can be omitted without any loss of information.

On receipt, the removed triangles can be regenerated from the information contained in the rest of the model. Since this method effects nothing but to introduce “well-planned” holes in the model, a good error-correcting package will be able to recoup the missing triangles, thereby obviating the need for a decompressing program.

This compression method can be improved further by requiring neighbouring triangles to contain a vertex, instead of an entire edge, for a specific triangle to be removed. Although much more computationally expensive and less robust, the compression ratio will be substantial.

One stands to derive the following benefits from this method, thus making it an important candidate to consider for STL compression:

- Completely loss-less compression is achieved.
- ASCII STL maintains the platform-independent format.
- Decompression can be effected by good error-correction software.
- Relatively easy to implement.
- Can be used in conjunction with other general and geometry-based compression methods to achieve high compression ratios.

7.3 Summary

The large size of STL files, especially that of the ASCII variety, complicates their transmission and storage. If good compression schemes were applied to

such files before their transmission, however, a dramatic reduction in transmission time could be effected.

Two categories of compression techniques are typically used to compress files, namely the general compression scheme used for all types of files and more specific compression methods that rely on imbedded geometrical properties to effect compression. Many of these compression schemes can, however, be combined to enhance the compression process even further.

The next chapter will be devoted to an overview of the types of errors that STL files are particularly prone to, followed by a closer look at each type of error.



Chapter 8

**An introduction to error checking and
the prototype**



8.1 Introduction

As was mentioned earlier, the building of a defective file should be avoided at all costs. STL files, however, are often flawed and a prototype built from an erroneous file will usually be worthless. For this reason, it is vital to check the file received and to verify that no errors have been created during its creation or transmission. This process could be effected either manually or, in the case of a large bureau, automatically [2].

The question whether or not STL files have been adequately checked and fixed remains a burning one and constitutes one of the biggest problems in telemanufacturing, especially since these files are sent from numerous unknown and, therefore, dubious sources. As the need for layered manufacturing arises in other fields, erroneous STL files will become even more problematic [2].

Errors and difficulties associated with the STL file format can be categorised into the following groups:

- Structural errors.
- Geometrical errors.
- Unmet building requirements.

This chapter will be used to determine the steps to be taken to check an STL file before it is sliced and finally built. Checking a file is a step-by-step process and each step must be taken to verify its correctness. Once an error has been uncovered, it must be corrected before checking could be resumed.

In the course of this dissertation, the author has not only explored existing algorithms for their ability to check and fix errors, but has also devised his own methods, where needed. The idea is to present these methods in a well-suited interface that will allow easy checking of a submitted file and which will generate a full report on such file, including recommendations on the object and

a list of errors that could and could not be fixed. This prototype and the areas it addresses will now be identified and introduced. A detailed discussion on the algorithms employed will follow in subsequent chapters.

The framework of the prototype can be depicted as follows:

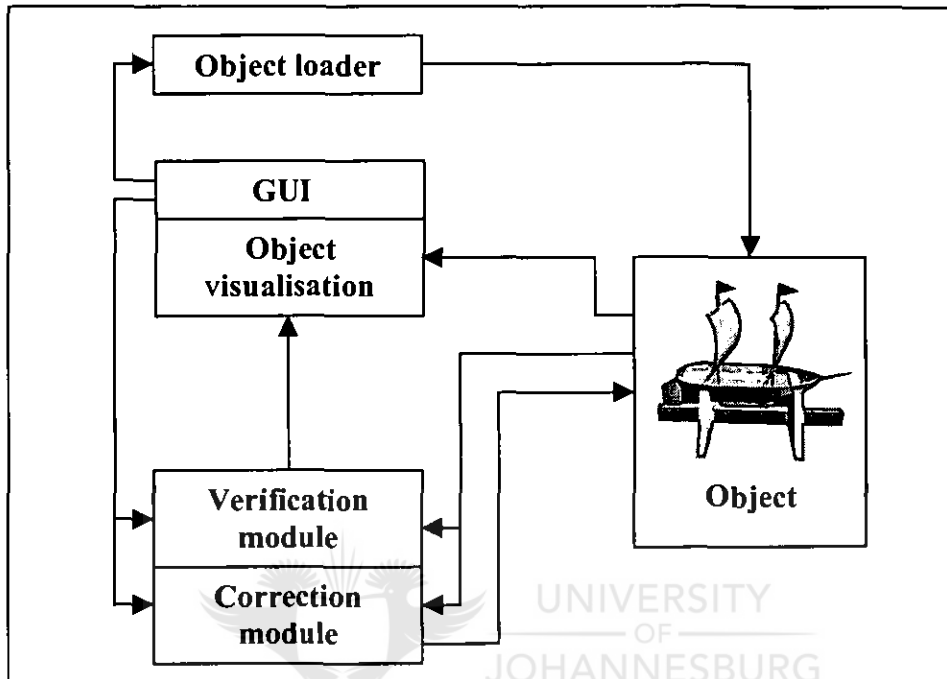


Figure 8.1: Framework of the prototype

The project consists of various modules, as shown above. The user interacts with the GUI to launch any of the other modules. From the GUI, the user can request a file to be loaded through the object loader. Once the object has been loaded and rendered through the object visualisation module, the user can request that the object be verified and/or corrected through the verification and correction modules respectively.

8.2 Principal aim of the project

The prototype was written for the Microsoft Windows environment, but the same algorithms could be applied to any platform. The project was undertaken with the following primary functions in mind:

- Detection and correction of common STL errors, including
 - file syntax and conformity to the STL format (for ASCII STL files)
 - checking for and removal of duplicate triangles
 - vertex-to-vertex checking and filling of simple holes
 - checking of triangle orientation
 - checking the file against Euler's rule for legal solids.
- Easy-to-use interface, thereby minimising training.
- Effective visualisation methods, including
 - rotation of object in x, y and z directions
 - translation of object
 - resizing of object.
- Visual error reporting, through the use of different colour codes.
- Full text report on an object.
- Support for both ASCII and binary STL files.

8.3 Error checking

Two sets of errors generally occur in STL files, both of which are addressed in the prototype. They are structural and geometrical errors.

8.3.1 Structural errors

This type of error usually crops up in ASCII STL files, although binary STL files could also contain structural flaws (for example, omitting the number of triangles after the header, as specified by the format). Structural errors can, in turn, be sub-divided into the following two categories:

a) File integrity

Flaws integral to the file itself are classified as structural errors. Although there is a wide variety of factors that could cause damage to a file, some have a higher incidence than others, such as bad transmission, poorly written software or even electronic sabotage by malicious parties. Some of these errors, such as reserved words being in the wrong case, are easily fixed, even though this type of error generally calls for human intervention.

b) Incorrect format processed as STL file

When a party sends a file to the bureau in the wrong format, the system should treat it as a special case of a structurally flawed file. When a file arrives at the telemanufacturing bureau, it needs to be verified to ensure that it is indeed in the STL format. Checking the file extension is definitely not enough to guarantee compatibility.

A scanner-parser pair was subsequently designed to address these two problems. This concept, adapted from the construction of compilers, allows the file to be analysed more thoroughly and generates more detailed reports.

Chapter 9 will be devoted to a detailed discussion on structural errors, whilst chapter 11 will be used to elaborate on the algorithms employed during the checking phase.

8.3.2 Geometrical errors

Geometrical errors have a high incidence in STL files, with the result that the files must be checked thoroughly [30, 2]. The following geometrical errors are commonly found in STL files:

a) Duplicated triangles in the model

This would represent a special case of vertex-to-vertex rule violation. The error is first detected by a procedure comparing every two triangles defined in the object. The following definition will allow us to define equal triangles:

For triangle **A** with vertices (A_0, A_1, A_2) and triangle **B** with vertices (B_0, B_1, B_2), let e be a pre-defined threshold.

$A = B$, iff there exists a unique ($0 \geq j \geq 2$) such that $|A_i - B_j| \leq e$ for each $i = \{0, 1, 2\}$. It is important that e not be chosen too large in order to prevent two closely spaced yet distinct vertices from being recognised as a single point.

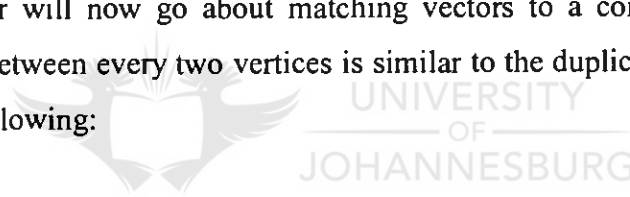
If two triangles were found to be equal, one of the triangles would simply be removed from the object.

b) Checking the data against the vertex-to-vertex rule

Each triangle in the object must meet all of its adjacent triangles in a common edge. This implies that each side must be shared completely by exactly two triangles. Failure to adhere to this rule usually indicates the presence of holes and zero-width walls in the model [21].

This rule is verified by first giving structure to the entire file. The vertices of each triangle are used to construct three vectors, which are stored in a linked-list structure. Every correctly shared edge in the file will, therefore, be listed as two vectors – equal in size yet opposite in direction.

The computer will now go about matching vectors to a common edge. The comparison between every two vertices is similar to the duplicated-triangle test, but for the following:



For vector **A**, with starting and ending vertices (**A_S**, **A_E**), and for vector **B**, with starting and ending vertices (**B_S**, **B_E**), let **e** be a pre-defined threshold and let the following hold:

If $|A_S - B_E| \leq e$ and $|A_E - B_S| \leq e$, then the edge would be correctly shared, as long as no other listed vectors shared it. If $|A_S - B_S| \leq e$ and $|A_E - B_E| \leq e$, then one of the vectors has the wrong orientation and needs to be changed. Apart from that, if no other vector shared the edge, then the edge would be correctly shared.

Should an edge be found that is shared by a single triangle only, the vector would be stored in a list for further analysis. After the entire file has been checked, the list of unshared vectors is examined in order to correct the file.

c) Ensuring that each surface normal do indeed point outwards

In the previous paragraph, a scenario was created in terms of which an edge is correctly shared by two triangles, but both vectors have the same direction. In such cases, one triangle is incorrectly orientated.

Every triangle must be checked against the right-hand rule to make sure that the orientation of the vectors be correct [10]. This is important for the calculation of each triangle normal. If a triangle were incorrectly orientated, the normal would point in the wrong direction [2].

By changing the concord of two of the vertices in the faulty triangle, the orientation of the triangle can be reversed.

d) Checking the model against Euler's rule for legal solids

For any convex polyhedron, the number of vertices and faces together is exactly two more than the number of edges. This meaningful result, discovered by Euler, allows an object to be verified as a legal solid [31]. Fortunately, the format of the STL file also conducts this check with relative ease.

The said check will ensure that the object does not contain any orphaned surfaces, zero-width walls, cracks or holes. An object that comprises a hollow volume will also fail this check. If an object were to fail the Euler-check, it would have to be analysed for anomalies.

The prototype also verifies this rule by counting the individual faces, edges and vertices, substituting these values in the equation and checking the final result.

8.4 The interface

The prototype incorporates a graphical user interface (GUI) that is not only user-friendly, but which also acts as a powerful visualisation aid, in terms of which information is conveyed to and from which commands are received from the user.

8.4.1 Visualisation

One of the principal aims of layered manufacturing is the visualisation and inspection of a physically manufactured object [8]. Computer visualisation, however, still is very important. If an STL file were rendered on a computer screen, the designer could still work last-minute changes to the file, just prior to its being built.

OpenGL¹ [32a] and DirectX² [32b], two mainstream technologies that are currently available, allow objects to be rendered relatively quickly, thanks to the interface they provide to the system hardware. The present prototype employs OpenGL for graphics rendering, thanks to its wide-spread use and support.

An important aspect of the interface is not only to render the given object, but also to allow the user to rotate it in any direction and to move to any arbitrary location on the screen. Resizing the module in real time is important too, especially for small and highly detailed models. These concepts were also explored in the prototype, specifically through the use of OpenGL.

8.4.2 Error reporting

In most cases, error reporting on STL files takes on the form of a textual file with numbers and statistics. The prototype makes use of various colour codes to indicate the errors on the object itself. The user can then decide how to correct the model. This feature is especially useful for uncovering minute cracks in the object, which are not always visible on the computer rendering.

A text report is also generated to complement the above scheme. Apart from the information imbedded in the file itself, it contains warnings where necessary and recommendations on errors that require human intervention.

¹ OpenGL is a registered trademark of Silicon Graphics, Inc.

² DirectX is a registered trademark of the Microsoft Corporation.

8.5 Complying with building constraints

Each LM machine has various deficiencies and requirements that must be supplied and met before a model could be built. Some categories of LM equipment require supports, while others do not. The maximum size and accuracy rating also differ from machine to machine and careful consideration must be given to whether or not certain types of equipment would meet the needs of a particular model.

Although this aspect of validation has not been explored in the prototype, it needs to be introduced here, since it forms an integral part of telemanufacturing. A detailed discussion on this matter will follow in a subsequent chapter. The most prominent building constraints are as follows:

8.5.1 Coordinate range of the model

Care must be taken that the object fall within the designated range. If this requirement were not met, the object should be rotated and/or translated to ensure that it does fall within the proper coordinate range.

8.5.2 Model size

Very large models cannot be built either. If a model were too big to be built in one pass, it should be adapted. One of the following two solutions should be applied here:

Scaling the object, so that it would fit in the building area. This method allows for a solid part, without any visible attachment. Not all applications, however, allow the prototype to be resized.

Another method would be to divide the object into various parts, which could be reassembled on completion. This method has the added advantage of ensuring that the original size of the model be maintained, even though the precise binding of the various parts may pose a problem.

8.5.3 Model orientation

The orientation of the model may affect various parameters of the model, as well as those obtaining to the building phase. These parameters include the

- precision of the model
- building material used
- time to completion.

Rotating the part before it is completed could have a dramatic effect on the foregoing factors. Depending on the requirement, rotation of the object should be effectuated to meet the user's needs.

8.6 Summary

Errors constitute a sizeable problem in STL files. This chapter was devoted to the introduction of such problems in more detail, as well as to a brief introduction on the prototype that has been developed to check these files for errors and to fix them, if any.

Building constraints was another aspect deserving of our attention. Because of these constraints, even an STL file devoid of any error would not necessarily be built successfully. As became evident from chapter 2, there are many different implementations of layered manufacturing and the same model cannot be built on all hardware. Size, orientation and model properties also play an important part in determining whether or not a model could be built on a specific device.

Structural-error checking, the first line of defence against errors occurring in STL files, will be discussed in the next chapter.

Chapter 9

Structural-error checking



9.1 Introduction

The text (also known as the “ASCII”) format is used when incompatible systems have to convey information over a communication medium, typically the Internet. A well-known example of this is the HTML format, universally used on the Internet to describe the content of a Web page [12].

Currently, there is a diverse collection of software programs on the market that does not create correct STL files [2] and in terms of which transmission errors are a frequent occurrence and issues such as security, electronic sabotage and incompatibility are still to be addressed. For this reason, it is very important to verify the syntactical structure of any received file to ensure that it complies with the STL standard. This chapter will, therefore, be devoted to a discussion on the scanner and parser pair that is employed to perform the latter function.

9.2 Verifying the syntax of the STL file

A scanner and parser pair works interactively on the STL file to ensure that the file

- is indeed an STL file
- conforms to the STL format
- is syntactically error-free.

The reserved words or symbols within the STL file are important for the correct functionality of the scanner and parser. The scanner will be responsible for identifying each of these symbols and for passing them through to the parser, which will then check the correct concord of these symbols within the file [33].

The list of recognised symbols is as follows:

- “solid” and “endsolid”
- “facet” and “endfacet”

- “outer”, “loop” and “endloop”
- “vertex”
- “normal”

Apart from these symbols, the scanner should also be able to identify positive floating-point numbers within the file, both in the normal decimal and in scientific E-notation. The decimal number 129.129 can also be written as 0.129129E+3 in E-notation and it is important to support this format, as various STL files contain numbers in this format.

Once the list of symbols has been compiled, the grammar of the STL file can be defined (shown here in BNF form):

```
digit = "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"
letter = "A", "B", ..., "Z", "a", "b", ... "z", "<space>", ".", ","
operator = "+", "-"
number = [operator] {digit} [ "." {digit} ] [scalefactor]
scalefactor = ("e" | "E") operator {digit}
ident = {letter | digit}

vector = number number number
vertex = "vertex" vector
triangle = vertex vertex vertex
outer_loop = "outer loop" triangle "endloop"
normal = "normal" vector
facet = "facet" normal outer_loop "endfacet"
solid = "solid" [ident] {facet} "endsolid"
STL-file = solid
```

Elucidation of symbols

[] : zero, or one instance of.

{ } : one, or more instance of.

() : normal mathematical definition.

| : choice between the element on the left or the right, but not both.

“” : reserved words or symbols shown within quotes (also shown in **bold**).

<space> : blank space at this position.

Once the grammar has been defined, the scanner and parser can be constructed from it. The principal aim of the scanner is to recognise each of the reserved words and identifiers in the file and to pass these symbols to the parser. Spaces, line breaks, tabs and other format characters are filtered out by the scanner and are not passed back to the parser, thus allowing the format to be more robust and completely independent as to its overall structure. A character or word not defined as a reserved word will raise an error during the scanning process.

The principal aim of the parser is to check whether or not the concord of the reserved words and identifiers passed back from the scanner conform to the STL standard. These syntax rules are laid out in the grammar of the file (shown above). This is achieved through a series of functions and procedures in a programming language that supports recursive programming, such as Pascal, C++ or Java. A function is written for every grammatical rule and subsequently called as and when required. If a reserved word or identifier were found to be out of concord, an error would be returned, otherwise the scanning and parsing process would traverse through the remainder of the file. Only then could one rest assured that the file is syntactically correct [34].

This method of syntax checking allows ready modification, should the format of the STL file change or improve, thanks to future research. The basic layout of syntax checking can be depicted as follows:

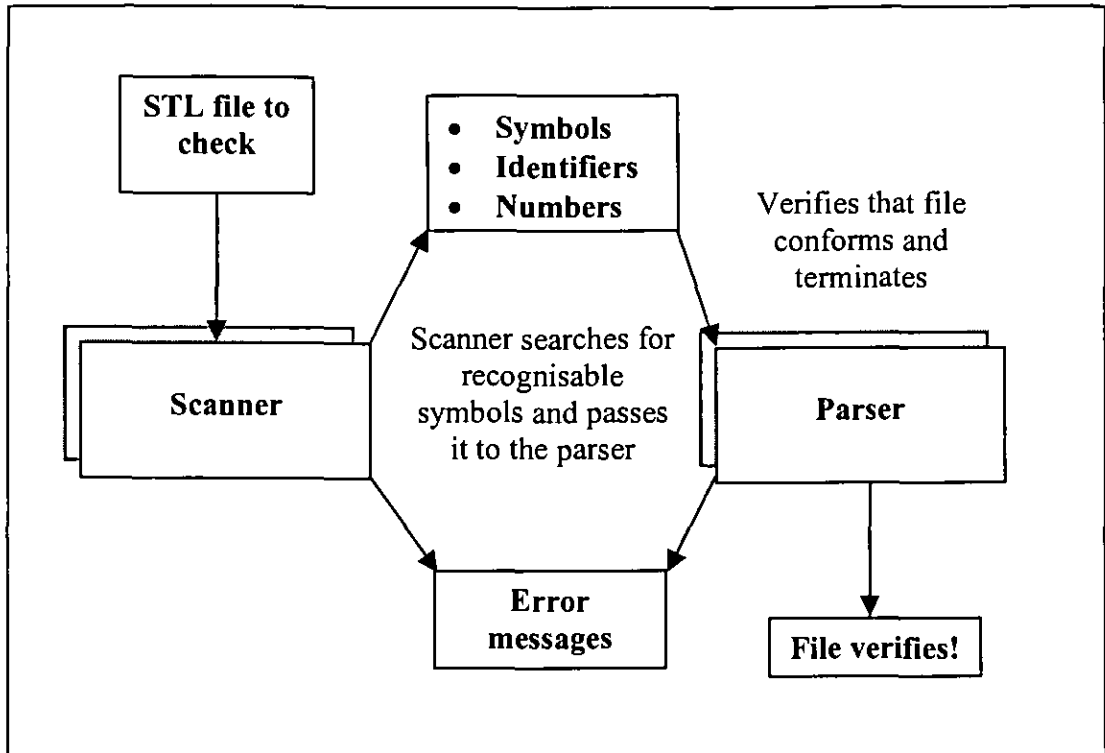


Figure 9.1: Layout of syntax-checking procedure

Once the syntax of the file has been validated, further checking can commence on the file. It must be stressed, however, that the syntax of the file must be 100% correct before any geometrical testing commences to prevent false error reports and damage to the file due to modifications by the software.

9.3 Fixing syntax errors

Errors in the syntax of text STL files will usually indicate either a file of the wrong format or a file severely damaged by software or any other means. Syntax-error correcting techniques have been successfully applied to various compilers in the past and can possibly be adapted for STL files too [34].

Some structural errors are very easy to rectify. A case in point here would be a reserved word in the wrong case. Although many software packages successfully read STL files in upper case, a package following the format very strictly may reject such files.

The parser will also indicate files in the wrong format. In such cases, the bureau might be able successfully to convert the format. The current approach to badly damaged files and unrecognised file formats, however, is to contact the client and to request that the file be transmitted again.

9.4 Summary

This chapter was devoted to the elucidation of the importance of syntax checking with STL files, as well as to the introduction of a highly effective method of doing so. The method explained in this chapter allows the bureau carefully to analyse a submitted file and to check that the file

- is indeed an STL file
- conforms to the STL format
- is syntactically error-free.

Once this has been achieved, further checking on the geometry of the model could commence, an aspect that will be explored in the next chapter.

Chapter 10

Geometrical-error checking



10.1 Introduction

Geometrical errors manifest on the object itself. Although the file describing the object may be correct, the object itself may suffer from a defect that needs correcting. Verifying and correcting such flawed STL files, however, is not only computationally expensive [29] but the testing process is also hampered by the numerical imprecision and redundancy associated with such flaws or defects [35].

Some geometrical errors merely involve the filling of gaps and the removal of duplicated triangles. More serious errors, however, may require integrate knowledge of the environment in order to be rectified, a process that would typically take place inside a rather complex expert system.

The present chapter will be aimed at providing an overview on errors that can easily be fixed by employing a set of algorithms, specifically designed for this purpose. In so doing, the prototype propounded in the previous chapter will also be scrutinised.

10.2 Removing duplicated triangles

The chances are that certain triangles may be duplicated during the creation of an STL file. This could also happen when a high-definition STL file is resized into a smaller version, which would cause some vertices to converge. Such duplicated triangles not only make for a bigger file, but may also impede the software slicing the file, ultimately giving rise to the creation of errors during the final building process.

A program checking the file against Euler's rule of legal solids may also have difficulty interpreting the extra face present in the file, reporting the solid as being invalid. Duplicated triangles in an STL file also violate the vertex-to-vertex rule (see the next section), requiring the removal of triangles, rather than the insertion of new ones.

It is evident, therefore, that it is imperative first to scan the STL file for duplicated triangles, which must be removed before any further checking could be effectuated. In terms of the propounded prototype, every two triangles are compared, the one with the other. If two triangles were found either to be the same or to fall within an acceptable threshold (as indicated in chapter 8), one of them would be removed. As the two triangles might differ in orientation, however, the correct triangle should be removed in order to avoid the introduction of a new set of errors to the file.

10.3 Checking the structure against the vertex-to-vertex rule

STL files are subject to an important restraint, namely that each triangle must meet all of its adjacent triangles in a common edge. This, in turn, implies that each edge be shared in full by exactly two triangles [3, 16].

An adjacent or edge-neighbouring triangle to triangle **A** is any triangle **B** that shares one of the edges of triangle **A**. This concept is important to the process in terms of which holes in the object are corrected and is graphically represented in figure 10.1. The adjacent triangles to triangle **A** have been marked **a'**. Those triangles bearing the letter **b'** constitute triangles that share a common vertex with triangle **A**, which will henceforth be referred to as “vertex-neighbouring triangles”.

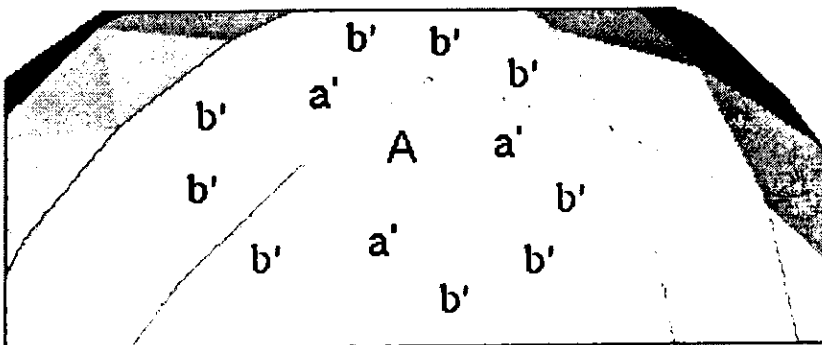


Figure 10.1: Adjacent triangles in an STL file

If edge-neighbouring triangles failed to meet along a common edge, the STL file would not be in compliance with the STL file standard. This error is one of

the most common errors in the STL file format and one of the most difficult to correct, especially in a region that has already lost several triangles [21].

In figure 10.2, five tessellated rectangles are shown. (Suppose, for argument's sake, that each of the images represents the top of a cube and that the borders of each rectangle are correctly shared with those of another triangle not shown in the diagram.) As becomes evident from these images, the three rectangles at the top of the diagram fail to adhere to the vertex-to-vertex rule, whilst the rectangles shown at the bottom of the figure are correct.

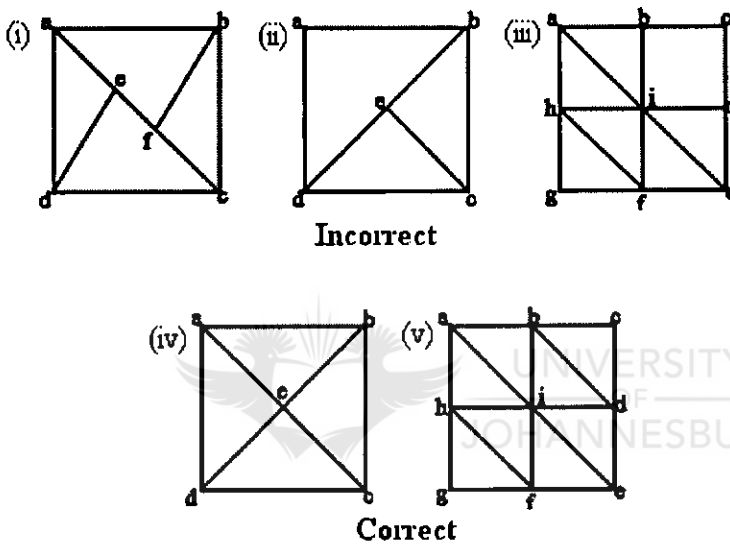


Figure 10.2: The vertex-to-vertex rule

In rectangle (i), four triangles are shown, namely **abf**, **bdf**, **dce** and **cae**. On closer investigation, it would become evident that edges **ae**, **ef** and **fd** are not fully shared between any two triangles and that they, therefore, fail to comply with the vertex-to-vertex rule. This is also true of rectangle (ii), which comprises three triangles (**abd**, **bcd** and **ecd**). In this example, edge **bd** is shared among all three triangles. This is not allowed either, however, since an edge must only be shared by two triangles. In rectangle (iii), edges **bc**, **cd**, **di** and **ib** are owned by one triangle only. These edges, in fact, define a rectangle themselves, namely **bcdi**, which is not allowed at all.

The rectangles shown at the bottom, however, are both correct. In rectangle (iv), triangles **abe**, **bce**, **cde** and **dae** each has one edge in common with another triangle. In this way, edge **ae** is shared between triangles **abe** and **dae**. This is in compliance with the STL file standard. The rectangle next to it has been tessellated at a higher resolution. It consists of eight triangles in total and also complies with the STL file standard. It is, in fact, the same rectangle as that shown in (iii), after the latter has been successfully corrected.

On closer investigation, it becomes evident that a legal solid will have three edges between every two faces. From this ratio, it becomes clear that, for a legal solid, the following rules must always obtain [2]:

- **F** must always be a multiple of two.
- **E** must always be a multiple of three.
- $3E$ must always equal $2F$.

The prototype propounded in this dissertation is aimed at checking for this error and at filling gaps that require the insertion of a single triangle. (The prototype does not, however, generate new edges and vertices.) Although it is very easy to check a file for this error, it is a time-consuming task that requires a fair amount of processing power to be performed within a reasonable time. The time span required is prolonged exponentially with an increase in the file size [29].

Checking and correcting an object against the vertex-to-vertex constraint requires a step-by-step procedure akin to the following (even though the proposed prototype follows this procedure, other packages may follow a different approach):

a) Step 1: Listing of edges

An STL file comprises a collection of triangles in a linear list, each consisting of three points in space in its turn. This architecture allows little checking

within itself and needs to be reorganised in a more suitable form in order to allow more intelligent error checking [29].

The three reference points of each triangle are used to construct three vectors, which are stored in a linear-list structure. Provided that the primary memory of the computer allowed it, the said list could be stored directly in memory, failing which a temporary file must be used for its storage. Storing the entire list in memory, however, is much better, since this would facilitate processing.

It is important to keep track of each of the vertices, as well as of the triangle to which each vector belongs. This is easily accomplished, however, since the number of vectors in the list would always be three times the number of triangles in the object. Should an error be uncovered, the coordinates of the points making up the edge would be needed successfully to correct it.

b) Step 2: Pairing of listed edges

After having listed the vectors either in a temporary file or in primary memory, the program can begin pairing these vectors. As was stated before, each edge must be shared by exactly two triangles. The next step would be to iterate through the entire list and to mark those vectors that share a common edge.

On completion of the foregoing step, all unmarked edges will reveal an erroneous section in the STL file.

c) Step 3: Listing of unshared edges

After having paired the edges, a list must be compiled of those edges that are not shared correctly between adjacent triangles. This list must then be analysed and used to correct the errors.

d) Step 4: Reporting to the user

During this phase, the program needs in a user-friendly manner to report to the user and state any errors that cropped up. The user can then decide if the program should correct the problem or if a new STL file should be created from the original CAD model (or whether it should be resubmitted, in case of telemanufacturing).

e) Step 5: Correcting errors detected in the file

When the vertex-to-vertex rule is not adhered to, it is usually indicative of a hole in the model, which needs to be filled (except in case of duplicated triangles, that is, in which case the extra triangle must merely be removed). It is important to add triangles in such a way that the geometry of the model would remain unchanged.

Fortunately, the neighbouring triangles contain enough information to reconstruct any gaps created in the object, provided that the number of triangles to insert is equal to the number of triangles omitted. In order to correct a model by inserting a single triangle in a specific region, a triangle requires all of its edge-neighbours to be present. In this way, no new vertices or edges are introduced into the model, since the neighbouring triangles are used to construct the new triangle.

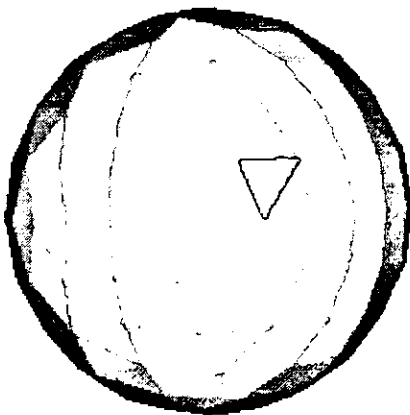


Figure 10.3: Approximated sphere with a missing triangle

Figure 10.3 illustrates just such a case. In this simple example, reinserting the triangle will result in the correct model. The three unshared edges make up the triangle that has been omitted from the model. Although there is no way in which to determine the exact number of missing triangles, three vectors sharing three common vertices most likely are the result of a single missing triangle.

The concord in which the vertices are listed in the new triangle is very important. If the concord were incorrect, the normal that accompanies the triangle would face the wrong direction and would introduce additional errors into the object. The direction of each vector of the new triangle should be the opposite of that of the vectors of the neighbouring triangles. This concept is illustrated in figure 10.4. Note the direction of the adjacent triangles, compared to that of the new triangle. (The new triangle is shown in the middle of the figure.)

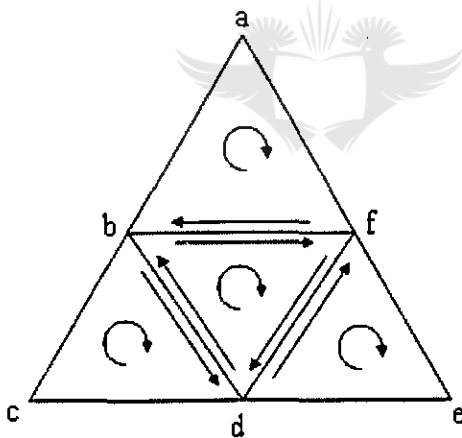


Figure 10.4: Triangle orientation of new triangle

When two or more edge-neighbours are missing, fixing the model becomes a lot more complicated. In such cases, new edges have to be generated in order to fill the gaps. When this is the case, the vertex-neighbouring triangles must be used to extract the information required to generate the new edges. The object could be fixed successfully, however, if at least one vertex-neighbour were present for every vertex. Special care must be taken, though, since more

than one triangle is now being inserted into the model. Edges must be correctly shared and triangles must have the correct orientation.

Difficulties will, however, arise when the program is faced with a hole in terms of which all of the edge-neighbouring triangles are missing and no vertex-neighbours are present to construct the new edges. It would be impossible to reinsert triangles now by using existing vertices of neighbouring triangles and it would be up to the program to “guess” what the object actually looks like. It would be better in most all cases if the file were re-created or resubmitted.

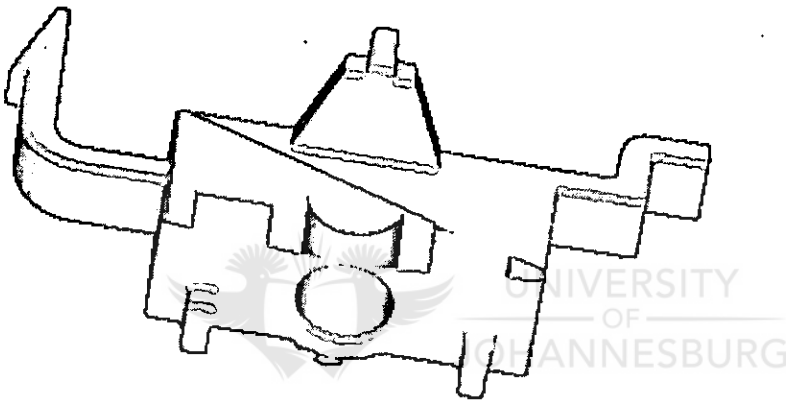


Figure 10.5: Badly damaged model

Figure 10.5 shows a damaged model that has not been repaired successfully, as several triangles are missing from the middle of the model. This model cannot, therefore, exist in the real world and cannot be constructed. (It is already difficult interpreting the model from the illustrated image.) Attempting to correct the error programmatically will result in disaster, owing to insufficient information in the remaining model. Human intervention will undoubtedly be required to repair the error.

Since there is no way to be sure that the file has been corrected, it should be inspected after correction to make sure that it does indeed describe the object to be built.

10.4 Ensuring that each surface normal do indeed point outwards

The surface normal that accompanies each triangle is used to distinguish the outside from the inside of the solid. The direction in which the normal is pointing indicates the outside. It is very important that the orientation of each triangle be correct and that each normal do indeed point in the right direction.

The direction of the surface normal is determined by the right-hand rule [10], which is nothing but the cross product between any two of the vectors defining the triangle. The length of the normal vector always is one [2].

Figures 10.4 and 10.6 are illustrative of the correct orientation of triangles, as well as of the incorrect orientation that will result in incorrect normal calculations.

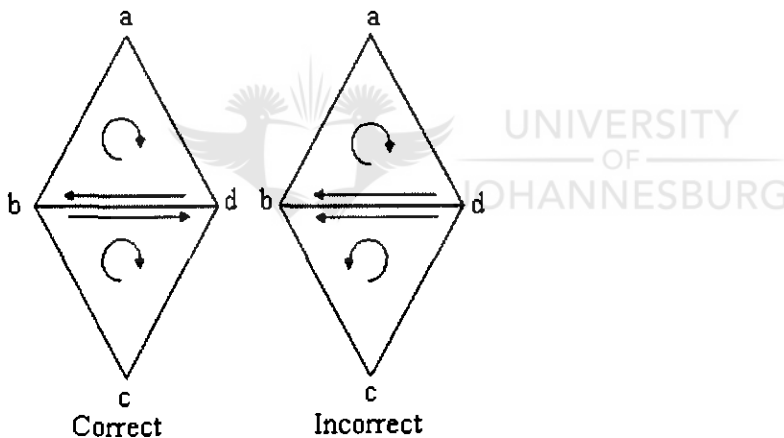


Figure 10.6: Correct and incorrect triangle orientation

The orientation of a triangle would be correct if each vector were to point in the opposite direction of the neighbouring triangle's vector, failing which one of the two triangles would have an incorrect orientation that would need to be changed. If all but one of the vectors were correctly orientated, then the adjacent triangle would most probably be the culprit.

When the incorrect triangle has been identified, the orientation can easily be corrected by changing the order in which the vertices occur in the triangle

definition. When the orientation of the triangles is correct, calculating the normal of each triangle is simple. This can be done by computing the cross product between any two vectors of a given triangle and by dividing each component by the vector length to obtain a unit vector. The resulting vector would be the normal and would indicate the outside of the model [36].

Although the present prototype is aimed at detecting this error successfully, it is not geared for the actual correction of this error in the model as yet.

10.5 Checking against Euler's rule for legal solids

For any convex polyhedron, the number of vertices and faces together is exactly two more than the number of its edges. For V , the number of vertices, F , the number of faces and E , the number of edges, this equation would be as follows:

$$V - E + F = 2$$

Proof for the above theorem now follows from induction [31]:

The solid can be viewed as a connected plane graph, say G . We employ induction on E . The base case, $E = 0$, $V = 1$ and $F = 1$, clearly satisfies the above equation. Assuming that the result be true for all connected plane graphs with fewer than E edges, where $E \geq 1$ and where G is supposed to comprise E edges.

If G were a tree, then $V = E + 1$ and $F = 1$ and the desired formula would follow. Alternatively, if G were not a tree, then let α be a cycle edge of G and consider $G - \alpha$. The connected plane graph $G - \alpha$ has V vertices, $E - 1$ edges and $F - 1$ faces, so that, by the inductive hypothesis, $V - (E - 1) + (F - 1) = 2$, which ultimately implies that $V - E + F = 2$, concluding the proof.

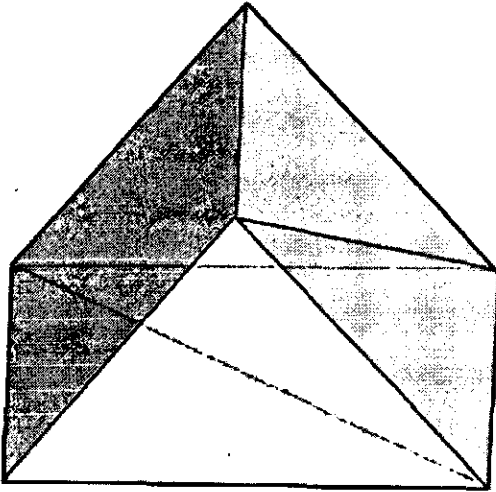


Figure 10.7: The prism

On visual inspection, the prism depicted in figure 10.7 is, for example, a legal solid (assuming that it is solid and that it does not contain any hollow interiors). The figure shows **6** vertices, **8** faces and **12** edges. Utilising Euler's formula for legal solids and substituting these values in the left-hand side of the formula yields $6 - 12 + 8$, which equals 2, satisfying the right-hand side of the equation. A model that fails to satisfy this equation might suffer from a geometrical defect, which needs to be addressed [2].

Since an STL file only comprises a number of triangles, each triangle consisting of three vectors and vertices, the task of checking for this rule is greatly simplified (although it could be a rather time-consuming one). Care must be taken not to count any edge or vertex more than once and duplicated triangles should be removed (see section 10.2 before commencing). The ease with which an STL file can be checked against Euler's rule of legal solids is a great advantage of the format.

Next, the proper procedure followed by the prototype to check a model against this formula:

a) Counting the triangles

The first step would involve the counting of the triangles present in the file. This number represents the number of faces in the solid. If no duplicated triangles were uncovered, this step would be greatly simplified. With binary STL files, the number of triangles can simply be gleaned from the file, since it is listed immediately after the header (the first 80 bytes of the file).

b) Counting the edges and vertices

Lastly, the number of edges and vertices is calculated. Each edge is shared between two triangles and each vertex is shared between multiple edges. It is, therefore, important not to count the same edge or vertex more than once.

c) Substituting and verifying

After having computed the number of faces, edges and vertices successfully, these values can be substituted in Euler's formula to verify that the solid would indeed be valid. Should the solid be invalid, the STL file would have to be recreated, since integrate knowledge of the environment would be required to fix the erroneous model and software might yield unwanted results. The defect might also originate from the original CAD model.

It is important in this respect to note that Euler's rule only obtains to solid shapes (any convex polyhedron). This means that a hollow object or any model containing an enclosed hole will fail this check.

Hollow objects are important, however, and have many characteristics that could be beneficial in certain applications. Some of these characteristics include

- less material to manufacture an object, thereby saving valuable resources
- a great reduction on some LM machines in the time required to build the object
- the option to alter the overall weight of the object

- the option to change the centre of gravity to some extent
- the possibility to allow applications that require an object with a hollow interior.

Unfortunately, LM hardware has been unable to create hollow objects to date. All objects containing peninsula structures require supports to enable building, which are manually removed after the object has been completed. It is impossible, therefore, to remove the supports enclosed by an object.

By splitting the model into several connectable pieces, however, a hollow model can be constructed. Each separate piece must meet Euler's condition and, after all the parts have been manufactured, the model can be assembled.

10.6 Summary

To prevent the LM hardware from constructing an erroneous model, an STL file has to be carefully checked for errors before building commences. Geometrical errors occur on the object itself and can be classified into the following groups:

- Removing duplicated triangles.
- The vertex-to-vertex rule.
- Inserting a single triangle.
- Inserting triangles by introducing new edges from existing vertices.
- Inserting new "guesstimated" vertices to create new edges and triangles.
- Triangle orientation and normal calculation.
- Euler's rule for legal solids.

Scanning the file before the building process commences will ensure that no time or material will be lost due to the building of a defective model. Although most critical errors can readily be detected, correcting them cannot always be effected successfully and may require the file to be recreated or retransmitted.

Chapter 11

Checking the file against building constraints



11.1 Introduction

Since the first LM device has been designed and implemented, the accuracy rating and time required to build a model have improved dramatically with every new model. Many new LM methods have been propounded and implemented successfully.

Each of these technologies, however, represents a set of new constraints against the actual STL file. A model designed to be built by one machine may not necessarily be successful on another. Depending on the actual hardware used, an STL file should be carefully analysed to ensure that it could be built on the specific LM hardware available [37].

This verification process can be divided into the following steps:

1. Ensuring that the coordinates of the solid fall within range.
2. Checking and possibly changing the object orientation.
3. Ensuring that the file be compatible with the hardware.

Before the object can be built, these constraints must hold. In some cases, the file can be manipulated in such a way that the model can be manufactured, but in other cases, human intervention is required. These aspects will now be discussed in more detail.

11.2 Ensuring that the coordinates of the solid fall within range

This step ensures that the actual position of the model in the file is such that the model can be constructed successfully. Various factors can affect this constraint, including the size of the model and the orientation of the part. Ensuring that the part fall within the specified bounds can, in turn, be subdivided into the following three steps or factors:

11.2.1 The size of the model

Each LM device has an upper limit regarding the size of the object that can be constructed. Before the building phase commences, the size of the model should be checked to ensure that the model would fit into the building area. If the model were found to be too big, one of two steps must be taken before building is attempted [10]:

a) Resizing the model

In some cases, the size of the model can be changed slightly to make it fit into the building area [10]. This constitutes a useful solution for models that are merely used for visualisation purposes. Changing the size of the model has the added advantages of curtailing the building time in which to construct the object and reducing the material and resources used.

Great care should be taken, however, when resizing a model, as certain parts of the model may become too small successfully to manufacture, with the result that the final model would be unsatisfactory. In such cases, the model should be revised or split into various sub-parts, which can be re-assembled afterwards. This step will be discussed in detail in the next section.

Advantages of resizing:

- It allows the manufacturing of large models on machines with a small building area.
- It does not require any additional post-building processing.
- It curtails overall building time and reduces materials and resources used to construct the object.

Disadvantages of resizing:

- The model no longer fits its original design size.
- Large designs containing precision areas may lose detail after having been resized.

- Resizing may cause vertices already in close proximity to each other to coincide, which will, in turn, result in errors.
- Resizing precludes the orientation of individual areas.

b) Splitting the model into two or more connectable parts

For some applications, especially for precision-critical parts, resizing the model is not ideal. In these cases, the only solution would be to split the part into two or more connectable parts, which parts can then be constructed individually by the hardware and assembled afterwards [10].

Although not always the ideal solution, splitting the object often is the most practical solution. If the manufacturer had access to more than one LM device, the parts could even be manufactured concurrently on each device, thereby reducing the build time of the entire object even further. By splitting the model into various parts, each part can also be rotated separately to maximise the accuracy rating of each individual part (the manner in and the extent to which the orientation of the part can affect the accuracy rating will be discussed shortly).

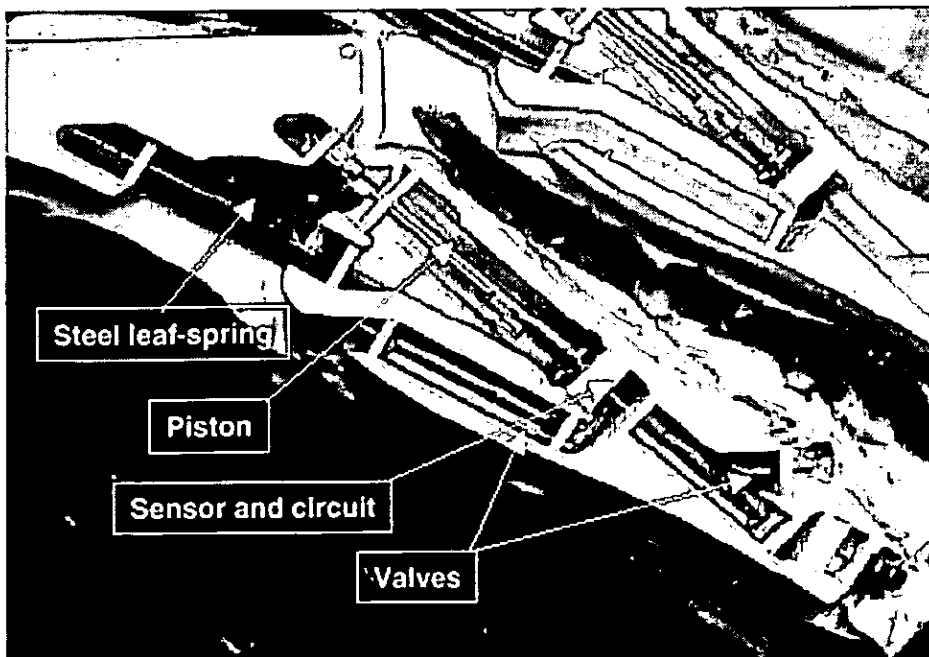


Figure 11.1: Robot leg after assembly [4]

When assembling the parts, the touching surfaces may need to be sanded to ensure that the parts fit neatly into each other. This step could be critical and care must be taken to ensure that none of the detail of the model be lost while the parts are being assembled.

In some applications, such as the robot leg shown in figure 11.1, the model consists of different materials and contains moveable parts, as well as electronic circuits. In such cases, it would be imperative separately to construct each part of the model. It is evident, therefore, that the splitting of an object into multiple parts constitutes a solution not only to size-related problems, but also to scenarios where electronic circuitry and various building materials are used in the same model.

Advantages of splitting:

- The model still fits its original design size.
- No detail loss to the extent of that in a resized model.
- Orientation of individual parts can be optimised.
- Parts can be constructed from various materials.
- Prototypes containing moveable parts can be constructed.
- Splitting allows for the manufacturing of complex, multisubstance models (see figure 11.1).

Disadvantages of splitting:

- Precise post-processing is vital when assembling the various parts.
- Splitting may prolong overall build time and may require more material (this is especially true if only one LM device were available).
- Splitting requires more calculations on the STL file(s).
- Not all objects can be successfully split into multiple components without complicating post-processing.

11.2.2 Distance from the table

The model should be elevated a certain distance above the building table by adding redundant building material between the table and the bottom of the model. These few layers act as a bridge, with the following purposes [10]:

- The layers serve to bind the object to the table.
- The layers facilitate part removal.

Without this bridge, the model could well drift away while being built. If, on the other hand, the model were bound to the table directly, the model would no doubt be damaged when removed afterwards. It is clear, therefore, that such bridge should be added to the bottom of the model.

The bridge is a physical part of the model and is added to the STL file during the design phase. Care should be taken that the model remain within the coordinate bounds of the hardware after elevation of the object.

11.2.3 Part orientation with respect to coordinate ranges

Changing the part orientation may also allow the object to fit in the build area, especially in cases where the dimension of the building area is non-symmetrical. A model that is too wide or too long can be rotated along the appropriate axes by a number of degrees, thereby ensuring that it would fit into the designated area [38].

This must be done with the utmost care, however, since the orientation of the model may impact on the building time and the accuracy rating of the object. This step will be discussed next.

11.3 Checking and possibly changing the object orientation

The orientation of a part has a marked impact on various aspects of the model, including the [38]

- overall accuracy rating of the model
- amount of building material used
- time required to complete the model [38].

Before a model is built, these parameters can be optimised by slightly changing the orientation of the part [2].

11.3.1 Part precision

In figure 2.4, the stair-stepping effect was illustrated when an arc is created by LM hardware. This effect can be greatly diminished by changing the part in such a way that all triangles lie either horizontally or vertically. This is, naturally, not always possible for all triangles. An optimum orientation will be one in terms of which as few as possible triangles are lying at a gradient, with the part still fitting inside the building area.

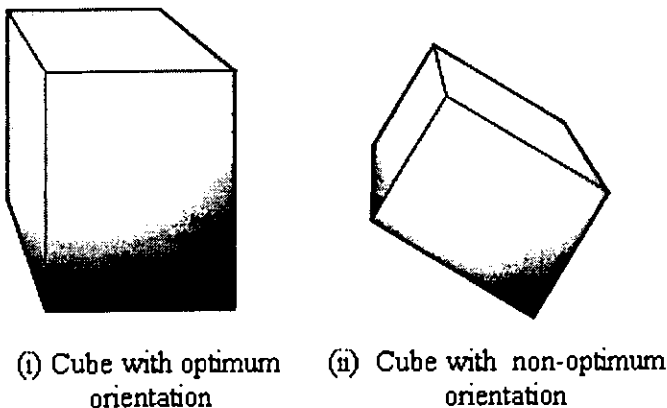


Figure 11.2: Orientation of a simple cube

Figure 11.2 (i) shows a cube that has been optimally orientated, with all its facets either horizontally or vertically aligned. The cube next to it, however, has no vertically or horizontally aligned facets and will suffer precision loss

owing to the stair-stepping effect when it is built. (It will, in fact, require support structures in order to be built!)

11.3.2 Amount of building material used

The amount of building material used in the manufacturing of a model differs from machine to machine and changes with each new orientation of the model. Certain technologies, such as stereolithography, require structural support to prevent the object from drifting away [1]. By changing the orientation of the part, the support required by the model can be greatly diminished, optimising the post-processing and also requiring less building material to complete the model [2].

In case of a technology such as Layered Object Manufacturing (LOM), in terms of which no support is explicitly included in the model design, the orientation of the model can also influence the total amount of material used. Here, the height of the object is directly proportional to the material utilised and can, therefore, be minimised by orientating the object to its minimum height.

11.3.3 Time to completion

It becomes evident, then, that if the material required to complete a model were to depend on the orientation of the part, it could also influence the time it takes to complete the model. Usually, the more material used in the model, the longer the build time [2].

In technologies such as stereolithography and FDM, orientating the model optimally can reduce the degree of post-processing required to remove supports from the object. The creation of the said supports during the actual building process will also add to the time it takes to complete the model.

In terms of technologies such as LOM, each layer of paper placed on top of the unfinished model takes time to cut and hatch and by maintaining an optimum height, far less building time will be required.

11.4 Ensuring that the file be compatible with the hardware

The last aspect that needs to be verified is whether or not the model can actually be built in the hardware available, since no single model could be built on all types of Layered Manufacturing equipment.

The size of the object should be verified to make sure that it would fit inside the designated building area. This issue has been addressed in detail earlier in this chapter. Some systems require support for ceilings and overhangs and if such support were lacking, the building of the model would prove impossible. The opposite, however, is also true, with the result that if a technology such as LOM were used to create the model, an object with imbedded support structures would needlessly lengthen the post-processing of the final model.

Secondly, the accuracy rating of the system should be checked to determine if the model could be successfully built and if it would, in fact, meet the precision requirements in question. Some technologies, such as LOM, are known to be incapable of rendering highly detailed models [8].

As some types of hardware are limited to specific ranges of materials, it is vital, too, to verify whether the material available would allow the building of the prototype.

11.5 Summary

After the file has been checked against errors, the model should also be verified to ensure that it be compatible with the available hardware. This is very important, since not all LM machines are fully compatible.

The following aspects should be carefully examined during this validation process:

- Object coordinates should fall within range.
- Object orientation should be optimally configured.
- The object should be compatible with the available hardware.

After the model has been verified, its building can finally commence. The next chapter will be used to elaborate on the prototype propounded in the present research study.



Chapter 12

The prototype: STLComplete



12.1 Introduction

STLComplete is a prototype developed for the Intel chipset, which is able to facilitate the visualisation process and, more importantly, the verification and fixing of both ASCII and binary STL files. The emphasis will be on those aspects that have been neglected in applications currently available.

Once again, the layout of the system, this time in more detail:

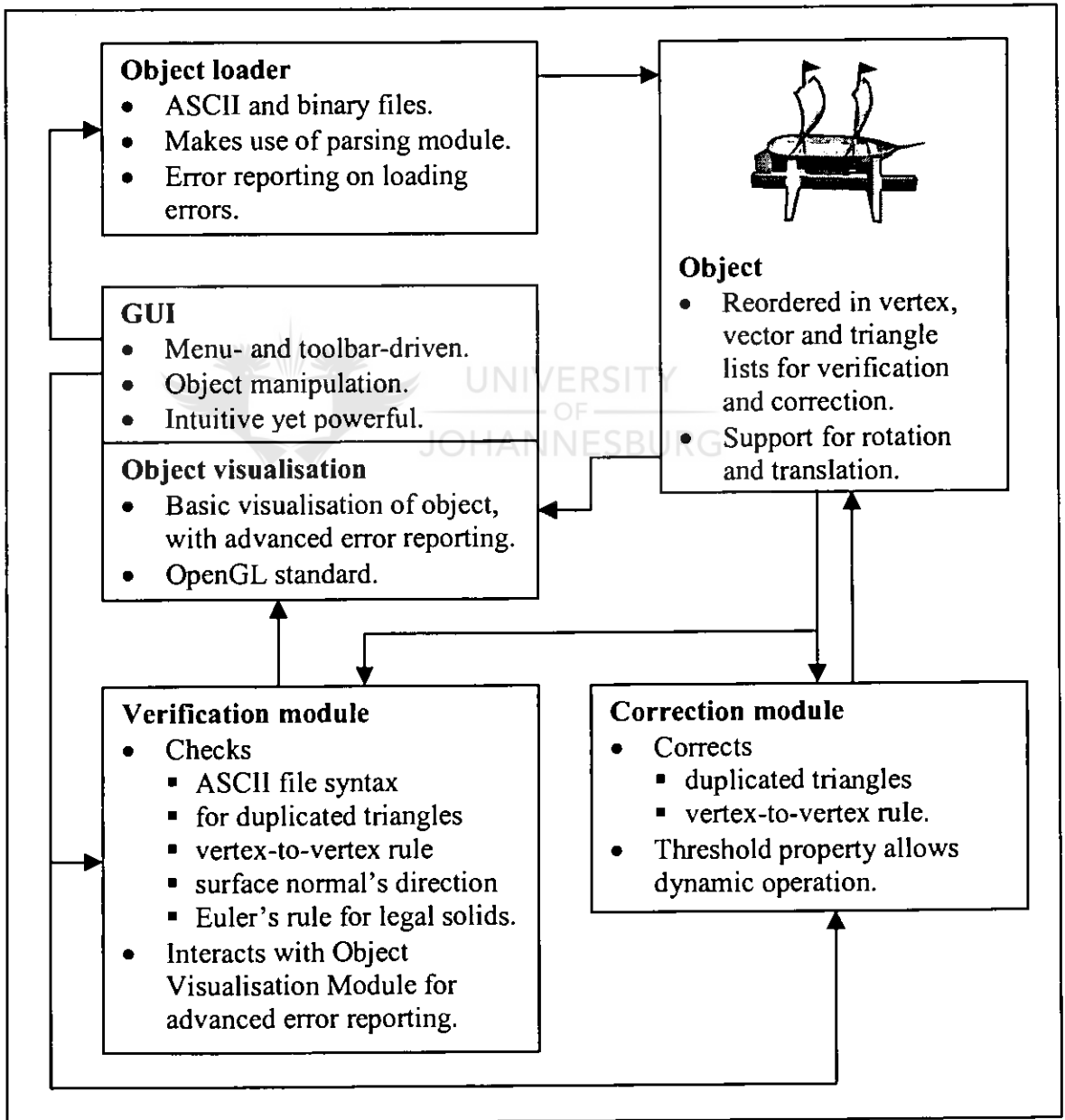


Figure 12.1: Detailed layout of the prototype

While chapter 8 was used to provide a brief introduction to the program, this chapter will be devoted to the algorithms used in the prototype, as well as to a more technical explanation on the workings of the program. The author will, in so doing, proceed on the assumption that the reader has acquired the necessary programming skills.

12.2 Syntax checking

The syntax-checking module of the program consists of two important parts that work in conjunction to check the syntax of the file. Together, these two parts, namely the scanner and the parser, form the basis of the syntax check [33].

12.2.1 The scanner

As was discussed in chapter 9, the scanner is responsible for removing any non-printable characters in the file. The basic layout of the scanner, as implemented in STLComplete, will be discussed here. For this part of the program, the following sets have been defined:

digit = {0, 1, 2, ..., 9}

whiteSpace = {non-printable characters such as spaces, tabs and line feeds}

reserved = {"solid", "facet", "normal", "outer", "loop", "vertex", "endloop",
"endfacet", "endsolid", "end"}

simpleOperator = {+, -}

procedure initialise(fileName of type string)

procedure shutDown()

The foregoing two procedures are responsible for initialising the scanner module. The *initialise()* function is called before scanning commences. This function opens and sets up the STL file and the counters necessary to keep track of the current position in the file, as well as of the next character to be read.

The *shutDown()* procedure is called after the scanning has been completed, in terms of which resources are released to the system. It also closes the STL file and releases the file handle to the operating system.

function getNextCharacter() of type character

This function starts from the first character in the file and returns the next character present in it. Once the end of the file has been reached, any further call will return the eof (“end of file”) character, indicating that the end of the file has been reached.

This is the only function in the entire scanning module that has any direct interaction with the STL file. Any optimisations that are directly linked to file access should be effected here. This includes maintaining a buffer to read a group of characters at once, instead of physically reading a single character with every consecutive call.

function getNextWord() of type string

This function mainly utilises the *getNextCharacter()* function and makes a number of successive calls to it. All characters returned are concatenated until a whiteSpace character is received. This whiteSpace character is then ignored by the *getNextWord()* function and the current word is returned to the calling function or procedure. (Any leading whiteSpace characters are also ignored and are not added to the word.)

function processNumber(testString of type string) of type boolean

When the first character in the string returned by *getNextWord()* is a digit or an operator, this function is called to determine the likelihood of the string being a number. The BNF notation of a number (which is widely used in the computer industry) can be written as follows:

scalefactor = (“e” | “E”) *simpleOperator* {*digit*}

number = [*simpleOperator*] {*digit*} [“.” {*digit*}] [*scalefactor*]

If the function were to identify the symbol as being a number, the function would return TRUE, failing which the function returns FALSE. The foregoing BNF notation is strictly adhered to during this procedure.

The following examples will clarify the difference between legal and illegal numbers:

Table 12.1: Examples of legal and illegal numbers in E-notation

Legal numbers	Illegal numbers
80.789	a9800.1
+27.67E-5	E56+5
-0.9999E+0	90E5
0.9212333E+6	0.9333+

function getNextSymbol() of type symbol

The foregoing functions are responsible for retrieving and verifying characters, words and numbers from the file. The present function uses these functions and identifies symbols from the file, categorising each and returning them to the parser.

The function commences by making a call to the *getNextWord()* function. If the word returned were a reserved word, the corresponding symbol would be returned by this function. If the first character of the word were a simpleOperator or a digit, the *processNumber()* function would be called with the word as a parameter to determine if the word were, in fact, a number. The process is repeated until the end of the file is reached. When this occurs, the scanner will return the eof (“end of file”) symbol.

The scanner will return a new symbol with every consecutive call to the *getNextSymbol()* function. It should be noted in this respect that the parser is built on top of the scanner and that it utilises the functions it contains.

12.2.2 The parser

This module now checks the concord of each identified reserved word and the correct use thereof. The main program also interacts with the parser by calling public functions from it. No direct interaction occurs between the main program and the scanner.

function foundVertexNormal() of type errorCode

This function is called when a vector or coordinate is expected, consisting of three consecutive numbers.

The function calls the *getNextSymbol()* function and checks if the returned symbol were a number. If this were not the case, an error would be raised, failing which the process would be repeated until three numbers have been successfully identified.

function foundFacet() of type errorCode

This function is called when the definition of a facet is expected in the file. It utilises the *foundVertexNormal()* function (defined previously) to analyse each facet of the model.

The function commences by ensuring that the next symbol be the “NORMAL” symbol. It then checks that the three numbers describing the normal are indeed correct. This is done by calling the *foundVertexNormal()* function.

Next, the program verifies the presence of the “OUTER” and “LOOP” reserved words respectively. The next step would be to verify the three occurrences of the reserved word “VERTEX”, each followed by a vector, again using the *foundVertexNormal()* function. Lastly, the delimiters “ENDLOOP” and “ENDFACET” are verified respectively.

Apart from checking each facet, this function also effects the loading of a text STL file into memory, if and when required.

function foundSolid() of type errorCode

The *foundSolid()* function calls the *foundFacet()* function for every facet described in the model.

This function commences by recognising the identifier that may contain a description of the file. This is followed by the triangles that make up the object. These are handled by the *foundFacet()* function and are called accordingly.

Once all the triangles have been identified, the function checks for the presence of the “ENDSOLID” delimiter.

function checkSyntax(filename of type string) of type errorCode

This lies at the heart of the parser module. This function gets called from the main part of the system and accepts the name of the file to be checked against syntax errors.

It is responsible for initialising the scanner and for returning system resources after the checking has been completed, making use of the *initialise()* and *shutdown()* functions respectively.

It identifies the first delimiter of the file, namely “SOLID”, and then calls the *foundSolid()* function. This function can easily be extended to support multiple solids within the same STL file, should the need arise to do so.

After the checking has been completed, the *checkSyntax()* function returns an error code, indicating whether or not the syntax check has been successful. A set of pre-defined error codes will indicate what type of error has occurred and where exactly.

The next section will be used to take a closer look at the module that effects the verification and correction of geometrical errors in STL files.

12.3 Geometrical checking

After an ASCII STL file has been verified free of structural errors, the file is verified against geometrical errors. (Binary STL files, on the other hand, do not require any structural checking.) Although greatly simplified, the following discussion will hopefully elucidate the basic algorithm:

The geometrical engine makes use of a set of lists, which enables the program to restructure the file into a more workable form. These lists are declared as classes and are accessed like normal variables. They are as follows:

TTriangleList, which is a list of triangles containing three vertices.

TVectorList, a list of vectors, each with a starting and an ending vertex.

TVertexList, which is a list of vertices, each with an x, y and z component.

Each of the above classes contains numerous member functions that assist in the management of the list. Once the STL file has been ordered in the above structure, specific algorithms can be applied utilising these lists in order to check for geometrical anomalies.

Should an error be uncovered in the object, all changes would be wrought directly to the list either by removing triangles or by generating new ones. Although some of the lists may be regenerated in the process, no changes are wrought to the actual STL file itself.

After the checking has been completed and the necessary modifications have been effected to the lists, the program generates a new STL file from the triangle list (kept in TTriangleList, as shown above). The only information directly used from the original STL file is the identification string found in the beginning of both the ASCII and binary STL files.

function isEqual(a, b of type object) of type integer

As this function is used in several of the geometrical tests, it will be discussed separately. The *isEqual()* function accepts triangles, vectors or vertices as parameters (although both parameters have to be of the same type).

The function returns an integer value, indicating whether or not the two objects are equal. A value of -1 indicates a difference in orientation with triangles and a difference in direction with vectors. (The value -1 has no meaning when vertices are compared and is, therefore, never returned.) If the two objects were found to be equal, the function would return a 1, failing which a 0 would be returned.

The function also includes a pre-defined threshold for each object type, so that vertices within this radius are deemed to be the same.

12.3.1 Removing duplicated triangles

The removal of duplicated triangles is a special case of a vertex-to-vertex rule violation, but it is separated to improve performance. The manner in which this error is fixed also is unique, since it does not require the insertion of triangles, but rather the removal of duplicated ones. The checking and correction of the file against this error is effectuated in terms of two separate functions.

function chkDupTriangles(T, E of type TTriangleList) of type boolean

This procedure is responsible for taking the entire set of triangles, **T**, and for returning a subset of this list, **E**, which, in turn, contains a list of all the duplicated triangles.

This checking procedure is written in such a way that the number of entries in **E** of a specific triangle is exactly one fewer than the number of instances of the same triangle in **T**. This means that if the same triangle

existed in **T** three times, it would only be listed in **E** twice (that is, two instances need to be removed).

The workings of this function is elementary yet effective. The program starts off by clearing and initialising the **E** list. Next, two indexes are defined, say i and j .

The first index, i , starts at triangle 1 and ends at triangle $n-1$ (for n triangles). The second index, j , is initialised to the triangle following the first index, therefore, $i+1$ and ends with $j = n$. The triangles at position i and j are compared. If found to be equal according to the `isEqual()` function, one instance of the triangle is added to the **E** list (for practical reasons, the triangle in position j is added) and the inside loop is terminated. The value of i is incremented and the process is repeated until both $i = n + 1$ and $j = n$.

After having effected the latter algorithm, the **E** list will contain a number of triangles that must all be removed from the **T** list.

function fixDupTri(T, E of type TTriangleList) of type boolean

This function uses the original **T** list, as well as the list of duplicated triangles, the **E** list.

The function simply starts off at the first triangle in **E** and promptly removes every triangle defined in this list from **T**. Since each triangle in **E** also stores the original pointer value of the corresponding triangle in **T** as a reference, the actual removal of triangles can be done very effectively.

12.3.2 Checking the object against the vertex-to-vertex rule

Compliance with the vertex-to-vertex rule is often violated and must, therefore, be carefully checked. Once again, the procedure consists of two separate functions, with the first function being used for the detection of the

error and the second function being performed to fix any errors uncovered in the object.

Although the first function serves to detect any vector not correctly shared, the fixing function addresses only those errors mainly caused by the omission of a single triangle. The function in terms of which to correct an object by inserting new edges and/or vertices into it has not been implemented in the present prototype.

function checkV2V(T, E of type TVectorList) of type boolean

This function takes the list of vectors, **T** (which has been compiled from the original set of triangles), and compiles a list of all vectors not correctly shared. These vectors are stored in the **E** list.

The function starts by initialising the **E** list and goes on to create an index that is initialised to the first vector in **T**. The program then searches for another vector in the object that describes the same edge. During this check, the direction of the vectors should be ignored. The program repeats this step for every vector in the list.

The same vector (vectors with identical starting and ending vertices) should not be found in two different triangles. Two vectors sharing the same edge correctly should have their starting and ending vertices reversed.

If the edge were not successfully shared between two triangles, the vector would be placed in the **E** list. Incorrectly orientated vectors will be ignored from here on and will not be added to the **E** list.

function isHole(a, b, c of type TVector, T of type TTriangle) of type boolean

In terms of this function, three separate vectors are compared to determine if they formed a triangle. If this were the case, the procedure would also be used to compute the new triangle and to calculate the corresponding normal.

The vectors of the new triangle are created in reverse order to ensure that the normal do indeed point in the right direction.

*function fixSimpleHoles(T of type TTriangleList,
E of type TVectorList) of type boolean*

This function takes the original list of triangles, **T**, as well as the set of erroneous vectors, **E**, as arguments.

The function calls the *isHole()* function with sets of three vectors to determine if the three vectors formed a triangle. If a hole were found, this function would be used to take the triangle returned by the *isHole()* function and to add it to the **T** list. The three vectors are then removed from the **E** list. This process is perpetuated until **E** is empty or until it contains vectors that do not form triangles. Any vectors left over in **E** are deemed to be indicative of the fact that the file needs to be retransmitted and that it cannot be successfully corrected by merely following the current correction scheme.

This procedure can be optimised by using a hash table to look for vectors within a specific range. By so doing, the need to search through the entire list to locate a specific vector is obviated.

The prototype only fills holes that require the insertion of a single triangle. A more complex algorithm can be implemented by searching for fully connected closed polygons and by then tessellating these polygons separately. Although this approach will fill the majority of holes, the

corrected model may not be what the designer had in mind at first. The program always follows the shortest route by inserting the smallest number of triangles needed to end up with a geometrically correct model.

12.3.3 Checking triangle orientation

Checking the orientation of each triangle is very similar to checking the object against the vertex-to-vertex rule. In terms of the latter check, we merely verify that each edge is shared between two triangles. With this test, however, we also verify that every triangle correctly follows the right-hand rule and we ensure this by checking that no two triangles have an identical vector, sharing the same edge.

function checkTriOrientation(T, E of type TVectorList) of type boolean

As with the previous procedure, this procedure takes the original list of vectors, as well as an empty list, which will be used to store these vectors that have been orientated incorrectly.

The program starts off by initialising the **E** list and the required indexes to traverse through the **T** list. When two vectors from two different triangles are found to be identical, both such vectors are added to the list. It should be noted in this respect, though, that each stored vector should contain a reference to the triangle to which it belongs, since this information will be required in order to correct the orientation.

After the entire list has been traversed, the **E** list will contain a list of vectors that could be incorrectly orientated. Unfortunately, not all of these vectors will have a wrong orientation, with the result that the correction of all instances of this error is no mean feat.

function checkEulerRule(T of type TTriangleList) of type boolean

Checking the object against Euler's rule for legal solids is a test seldomly included in software packages. Any object, however, has to pass this rule

in order to be successfully built on any LM machine. If an object were to fail this check, something would invariably be wrong and human intervention would be required. This error would not necessarily be uncovered by any of the previous checks.

The procedure is provided with a list of the triangles in the object, from where the following is carefully calculated:

a) The number of faces

The number of faces, in other words, the number of triangles in the object, is computed next. We can now proceed on the assumption that there are no more duplicated triangles (especially since all duplicated triangles have been removed with the first check already) and that a mere counting of the triangles in the **T** list will suffice.

b) The number of edges

The number of edges is computed next. Since the same edge is shared between multiple triangles, the procedure will first verify that the current vector has not been added before incrementing the counter.

c) The number of vertices

The number of vertices is computed in the same manner as the number of edges, taking care that no vertex is counted twice.

After the number of faces, edges and vertices has been computed, the procedure will calculate the value according to Euler's formula and report back to the user.

12.4 The interface

One of the principal aims of this project was to create an intuitive and user-friendly interface that will allow the user the necessary flexibility and adaptability when inspecting a model.

When the program is initially loaded into memory, the user is presented with a blank screen and a menu, from where a file can be chosen to be loaded. As was stated before, STLComplete supports both the ASCII (or text) and binary STL formats.

The process in terms of which a file is loaded, is graphically represented in figure 12.2 below:

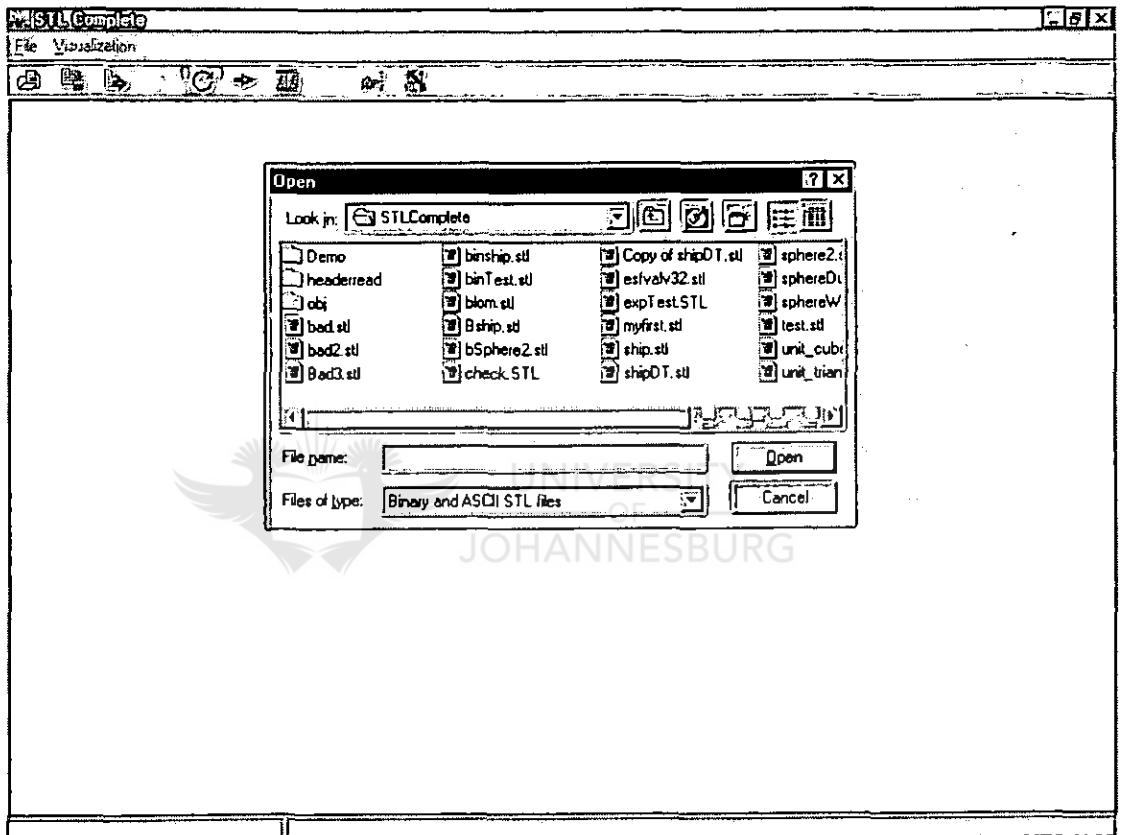


Figure 12.2: Selecting an STL file to load

Once a file has been selected and loaded into memory, the program renders it on the screen. Once rendered, the user can rotate, translate and scale the object to meet his/her needs. This flexibility allows the user visually to verify the object to some extent. The user can also select that the object be viewed as a solid, as a wireframe or as a set of vertices. The former two options are graphically depicted in figures 12.3a and 12.3b respectively.

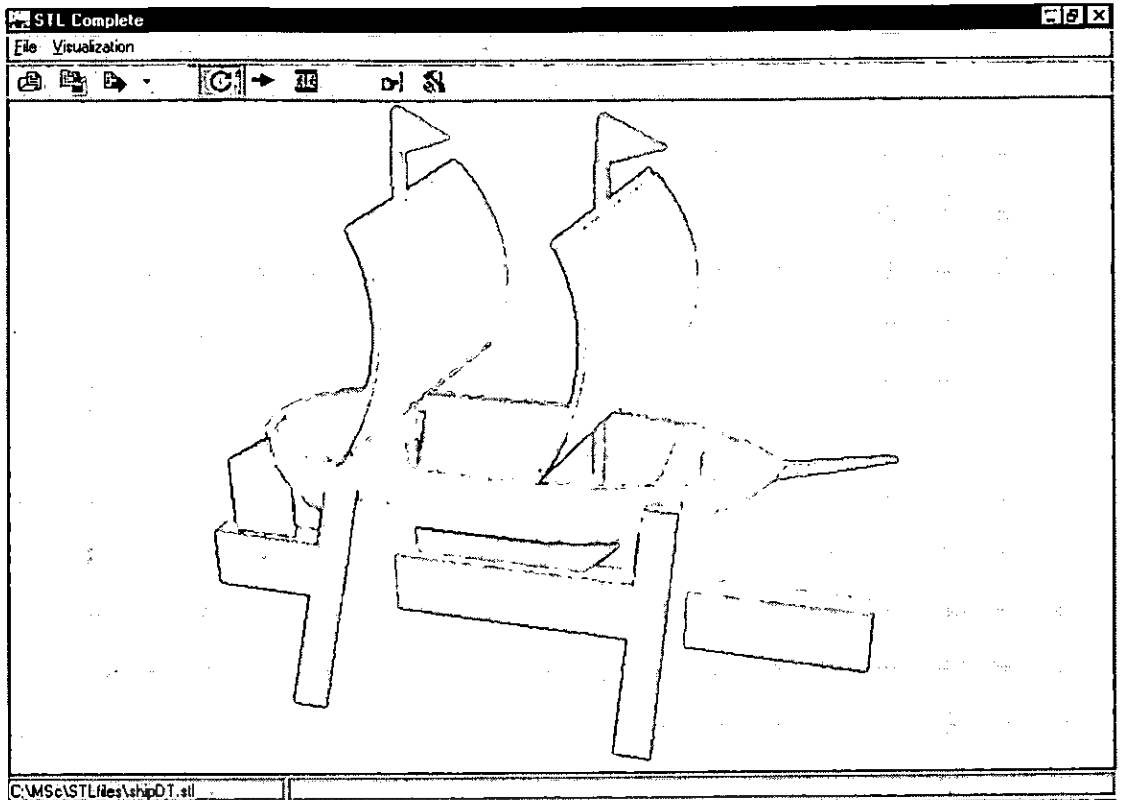


Figure 12.3a: Viewing the model as a solid

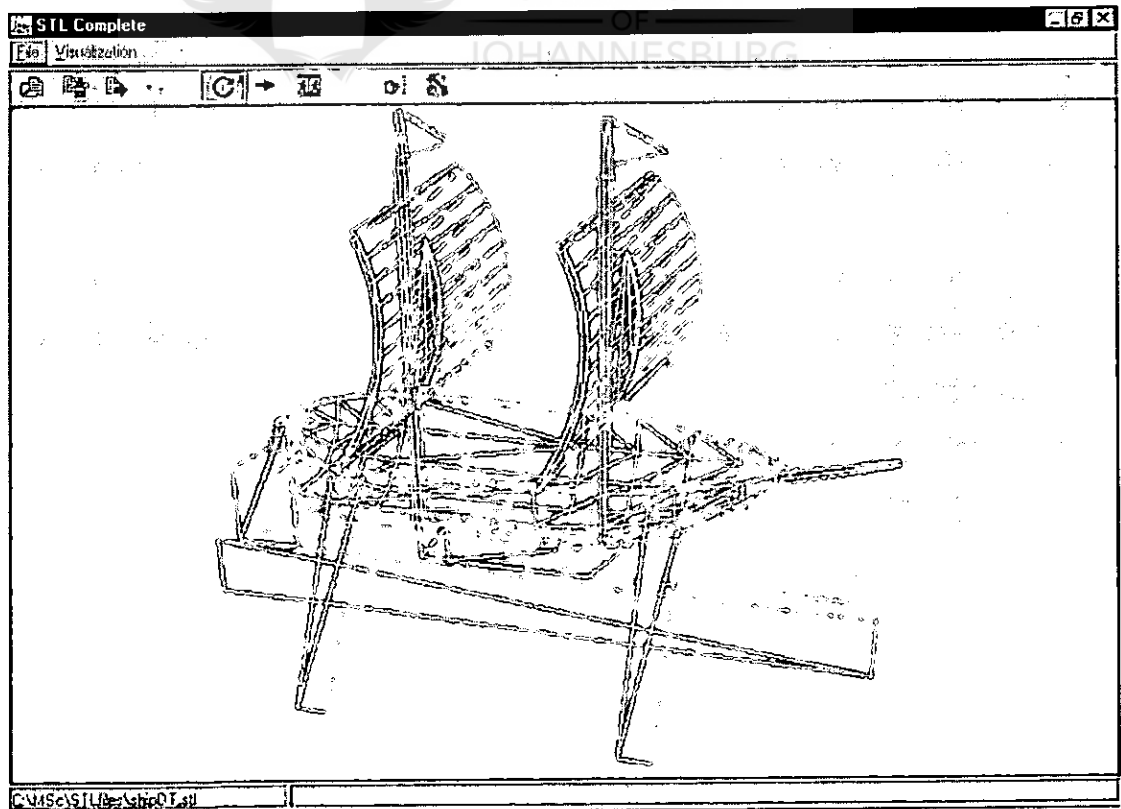


Figure 12.3b: Viewing the model as a wireframe

Next, the part can be checked for anomalies either by selecting the “validate file” option from the menu or by selecting the option from the toolbar. A progress indicator at the bottom of the screen will show the progress made with the testing function. Once the checking procedure has been completed, the object is rendered again and errors in the object are indicated by a spectrum of colours indicating the various errors in the object. Duplicated triangles will be marked red, a violation of the vertex-to-vertex rule will appear green and all orientation errors will show up blue. Following, a graphic representation of a badly damaged file, as depicted in figure 12.4, after checking has been completed:

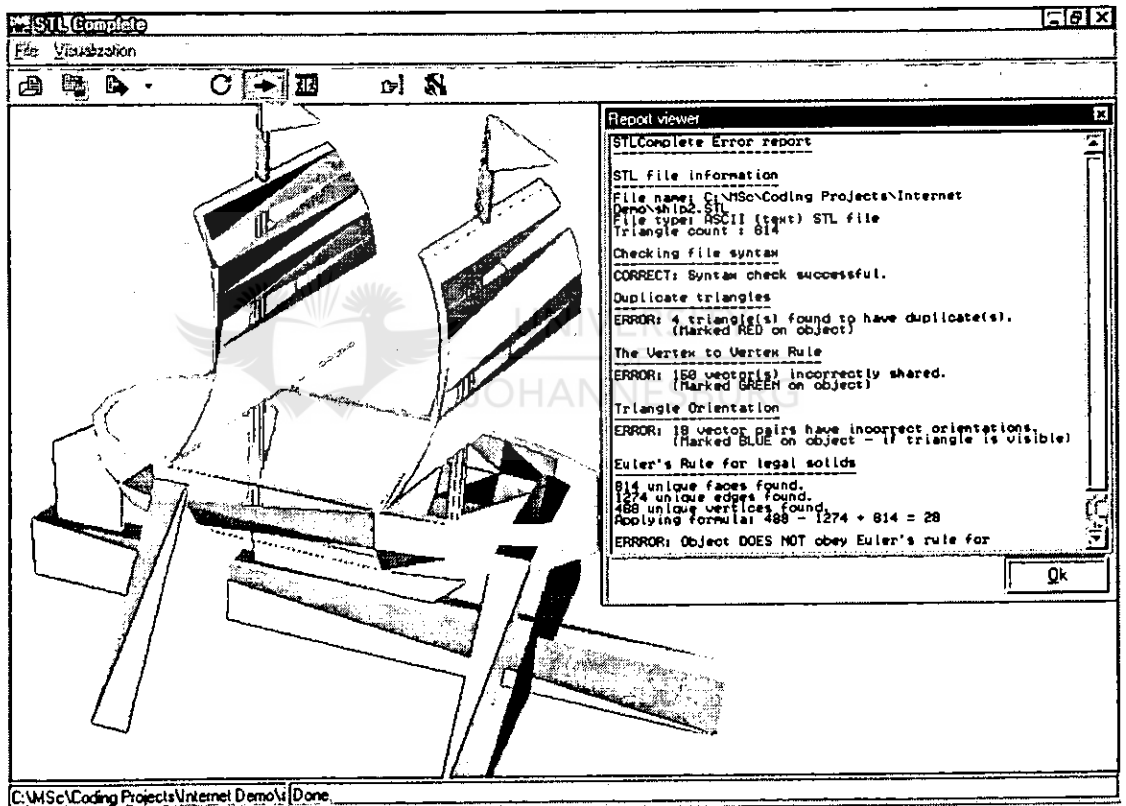


Figure 12.4: A badly damaged model, with errors showing up in various colours

As depicted in figure 12.4, a textual error report is also generated during the verification process, describing the uncovered anomalies in detail.

The user can exercise one of three options during the validation process. He/She can opt merely to check the file, ask for a prompt on each error or require the program to check and correct the file automatically. These options

can be exercised by selecting “options” under the menu or by pressing the corresponding button on the toolbar.

The user will then be presented with a screen, which will allow the threshold to be changed and the selection of the required error-checking mode. This is demonstrated in figure 12.5a. Here, the program has loaded a sphere with obvious flaws. The user selects the “automatic repair” option, presses the “okay” button and selects “validate file” from the menu. As was depicted in figure 12.5b, the object has been successfully fixed and can now be built by the appropriate hardware.

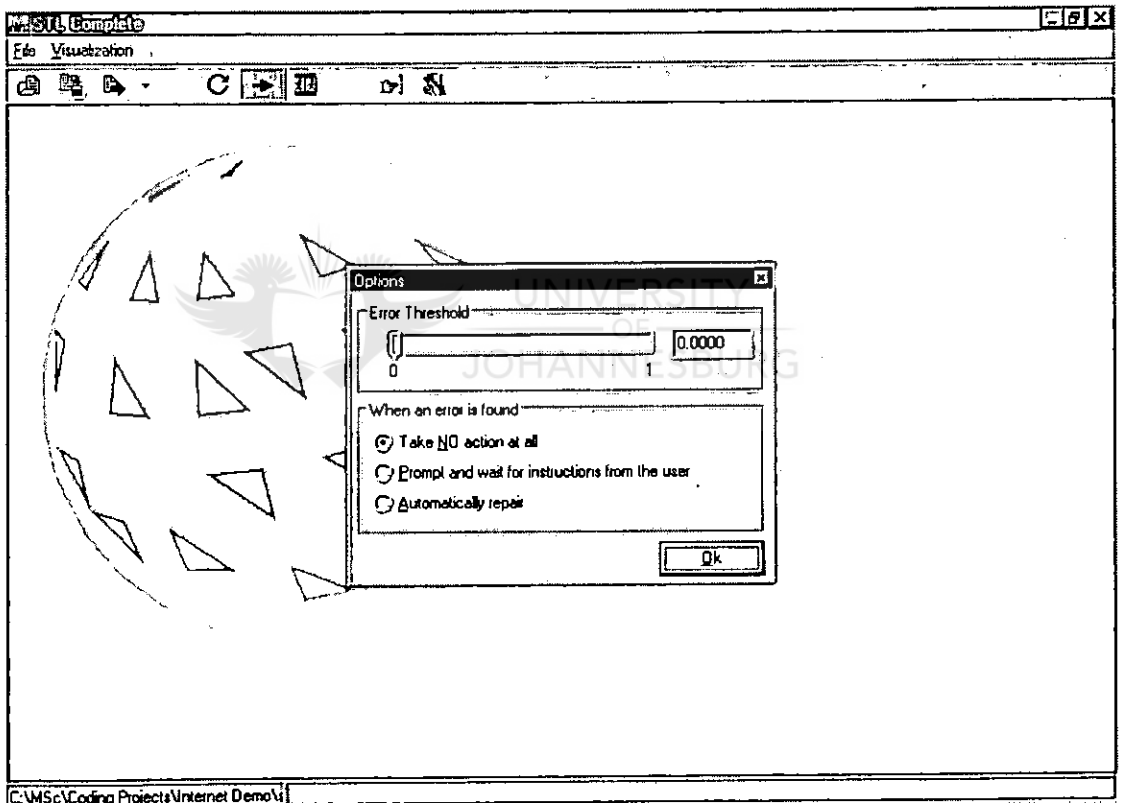


Figure 12.5a: Fixing the model

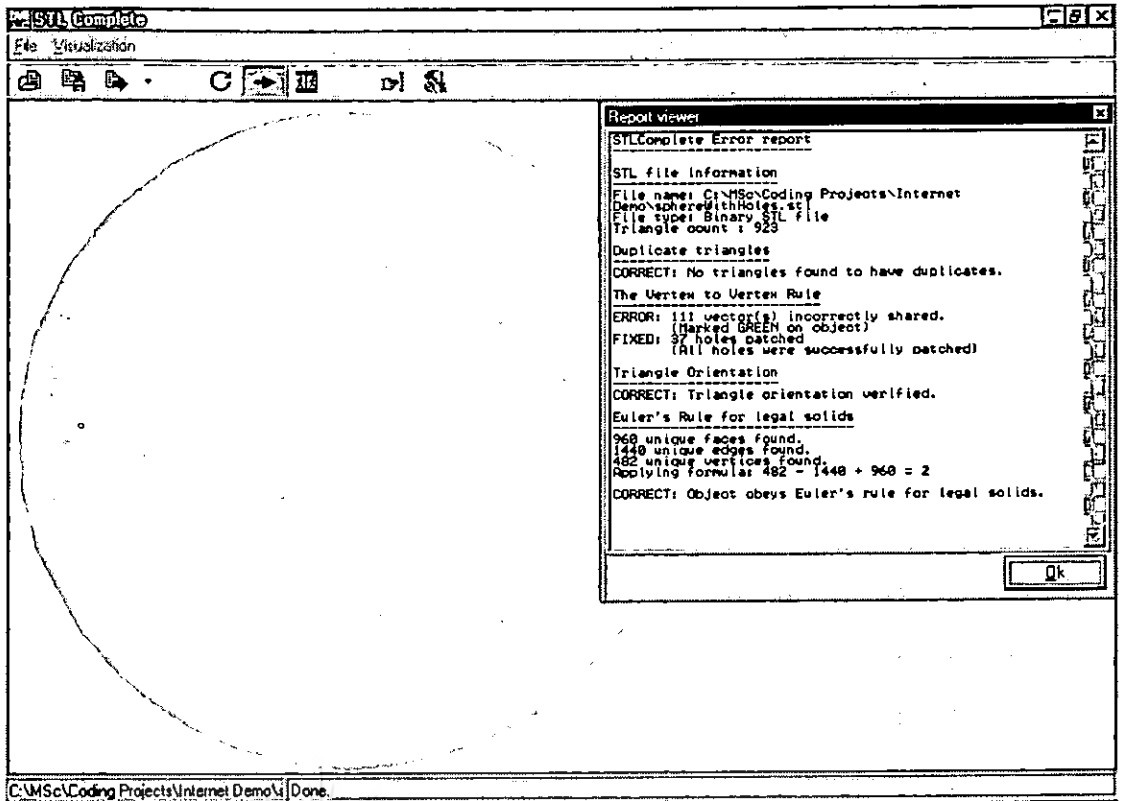


Figure 12.5b: Revised model

12.5 Summary

This chapter was devoted to an overview on the project propounded in this research study. Some aspects, such as visualisation and most of the error handling, have, however, been dealt with in other software packages.

Although some of the issues addressed in STLComplete are new, they have been shown to be important and their integration with certain software packages has been shown both possible and practicable. These issues include

- the checking of the model for duplicated triangles
- the checking of the object against Euler's rule for legal solids
- error visualisation by means of certain colour codes.

Many other software packages have also been analysed for the purposes of the present study, however, and the next chapter will be devoted to a brief discussion on the application of each of these packages.



Chapter 13

Software comparison



13.1 Introduction

This section of the dissertation will be used to review several software packages that are currently being used in the industry to manipulate STL files and to correct errors as discussed in previous chapters. Most of the packages to be discussed for this purpose support numerous file formats, in addition to the STL format.

In this discussion, emphasis will fall on the intelligent side of each software package, which mainly hinges upon its ability to uncover and correct errors in a solid.

13.2 STLview version 7.0a

STLview is a Microsoft Windows-based program that allows one to analyse, fix and embed or link 3D solid models to documents. Designed and coded by Igor G. Tebelev, it supports many file formats and allows the user to import and export a model from and to any other format of his/her choice. The package is released as shareware and is available for downloading from various sites on the Internet. Although some of the features have been disabled, they will be enabled on registration of the package [39].

13.2.1 System requirements

The minimum system requirements, as stated in the documentation of the product, are an IBM PC AT486 or higher processor, with an 8-bit colour depth (256 colours) display and a resolution of 640x480 running Microsoft Windows 95/98 or Windows NT [39].

After evaluation, it became evident that a fast Pentium-based system would be appreciated when some of the more processing-intensive features of the product are utilised. This includes the "playing around" option and the ability to fix errors in the solid [39].

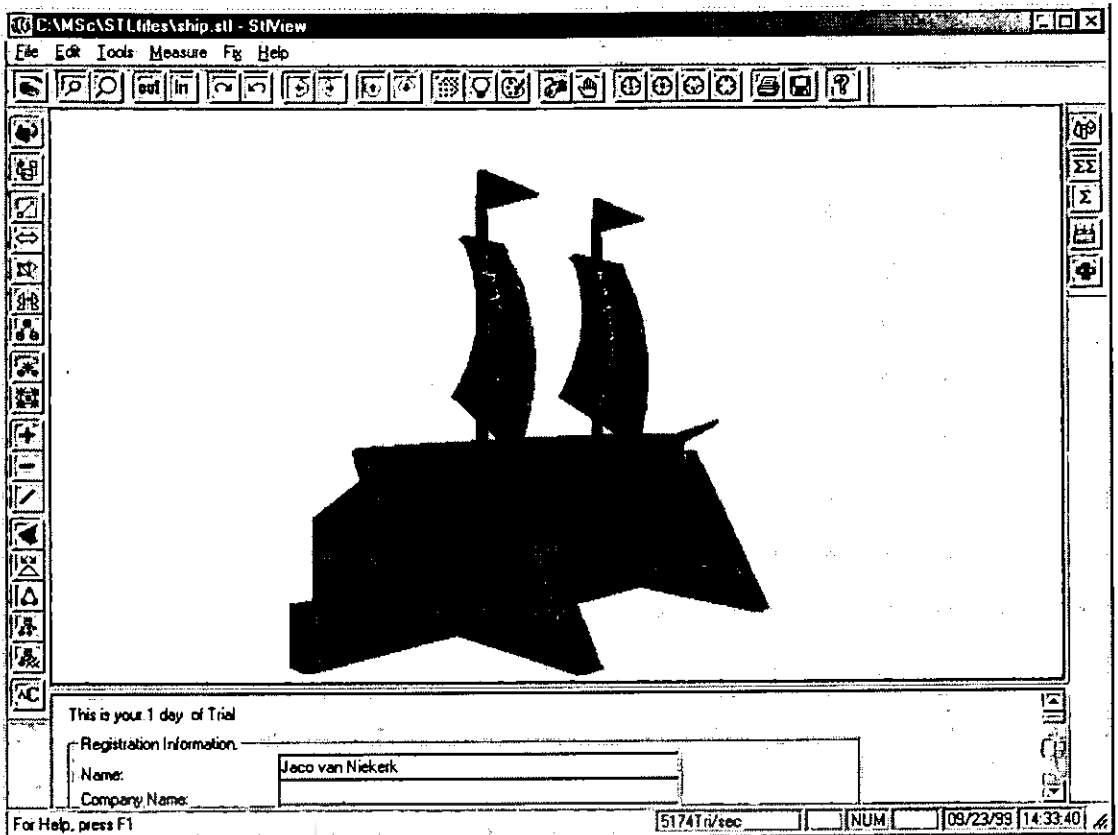


Figure 13.1: STLview interface (SDI option)

13.2.2 Program features

a) Interface type

As depicted in figure 13.1, the interface is highly interactive and allows real-time rotation, scaling and translation in any direction that the user requires. An option to zoom into the solid is also supported, which allows the user more closely to inspect the finer detail of the model [39].

The user is presented with a choice of two different kinds of interfaces, namely a Single Document Interface (an SDI) and an interface that supports a pop-up menu system [39].

b) File types supported for import

- ASCII and binary STL format
- 3DS format

c) File types supported for export

- **STL** format
- **DXF** format
- **IGES** format
- **3DS** format

d) Error correction supported

- Fixing of gaps and topologies of objects
- Correcting triangle orientation

e) Salient and added features

- Measuring of the volume and surface area of solids
- Merging of a number of solids
- Boolean operations on solids
- Design protection in embedded documents

13.3 Materialise Magics RP 4.3

Materialise Magics RP 4.3 is a professional STL file verification and a correcting tool. Its basic interface layout is depicted in figure 13.2 [40].

This software package has been released by Materialise and is useful for general manipulation and error correcting of STL files. The package is released as shareware and requires registration after an evaluation period for further use [40].

13.3.1 System requirements

The documentation supplied with the product suggests an IBM Pentium-based CPU with 32 Mb of primary memory running Microsoft Windows 95/98 or Windows NT as an operating system [40].

Although this software supports real time rotation and translation of objects, a more powerful system will be necessary in more complex solids for smooth operation [40].

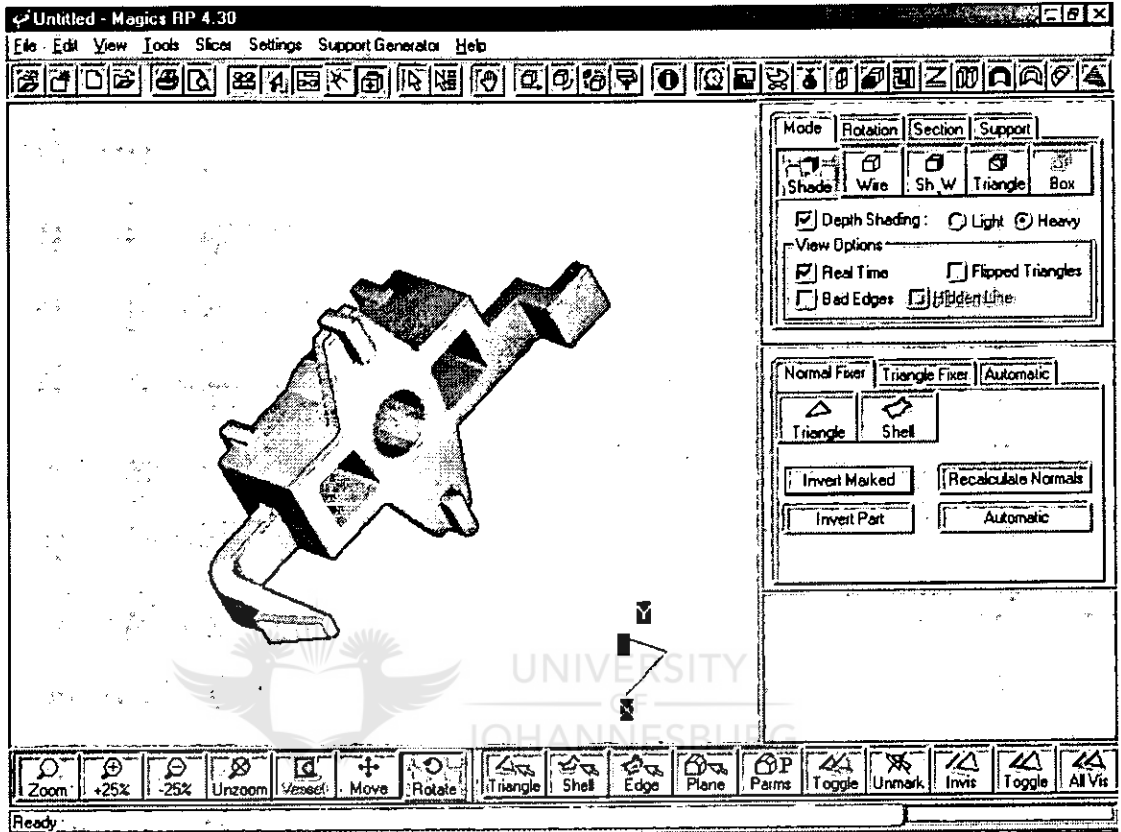


Figure 13.2: Materialise Magics RP 4.3 interface

13.3.2 Program features

a) Interface type

The product utilises a well-designed and intuitive user interface that makes it easy to manoeuvre between the functions that the package offers [40].

A drop-down menu lists all the tools that the user may require. A handy toolbar is situated beneath the menu for more ready access to commands used more often. Objects are loaded and placed on a canvas, from where the user can rotate and translate them as required [40].

b) File types supported for import

- ASCII and STL format
- IGES format
- VDA format
- DXF format

c) File types supported for export

- ASCII and binary STL format
- VRML format
- DXF format

d) Error correction supported

- Filling holes and stitching
- Correcting the orientation of normals
- Manual error correction

e) Salient and added features

- Calculating building time on LM equipment
- Inserting of simple pre-made objects
- Distance/Radius/Arc and angle calculation
- Visual error report
- Slicing with visual preview



13.4 STeaL version 1.2

STeaL is a very simple STL utility that is distributed free of charge from the Internet. Although the program does not support any error correction, it is useful for the visualisation of many 3D formats, as well as for exporting between these formats [41].

13.4.1 System requirements

The STeaL software does not include any system specification with the software and it is up to the user to discover the optimal system configuration

required by the package. The software makes use of Microsoft's DirectX library and it must, therefore, be installed on the system [41].

The program does not make use of intensive operations and any Pentium-based system with hardware-accelerated video capabilities should suffice in the use of this application. Even an 80486-class machine would be enough for small, simple models [41].

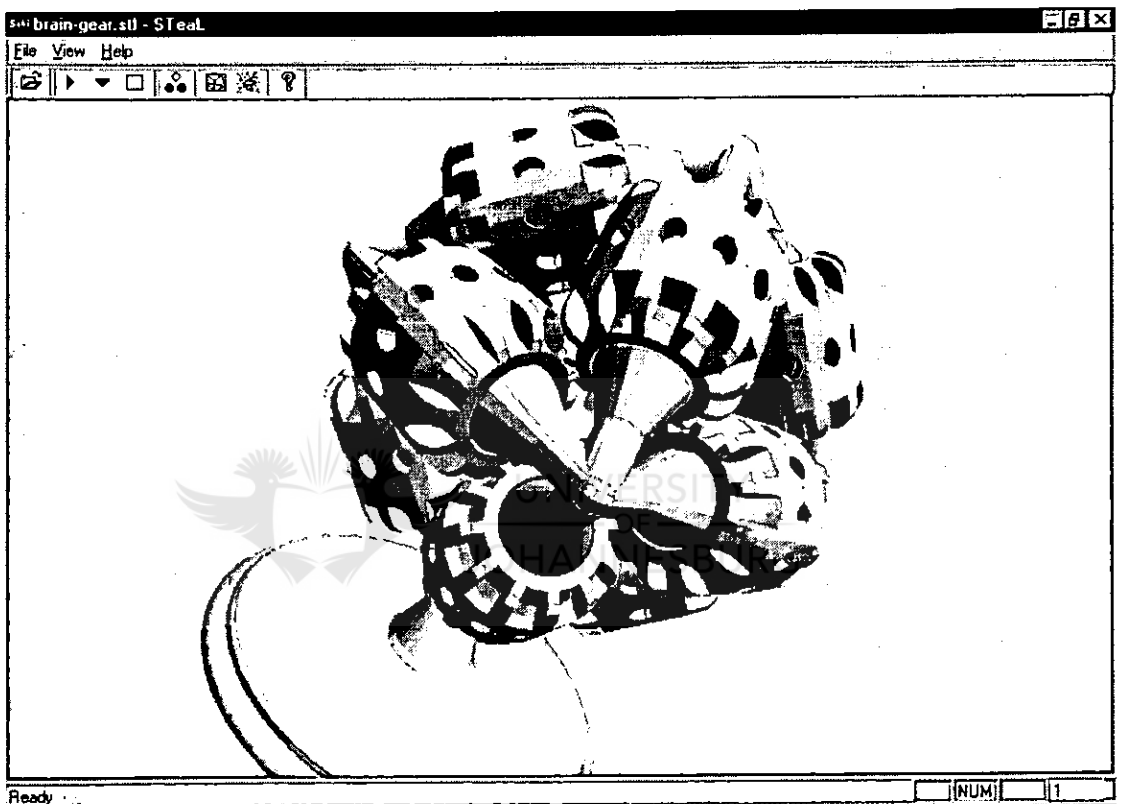


Figure 13.3: STeal by CIP software

13.4.2 Program features

a) Interface type

An intuitive and easy-to-use interface is supplied to the user. The user conveys commands to the system through a toolbar at the top of the screen or by using the menu system. The image is displayed in the centre of the screen, from where it can be rotated in any direction or scaled to the required size [41].

b) File types supported for import

- **ASCII and binary STL**
- **TRI** format
- **OBJ** format
- **RAW** format
- **LWO Lightwave** format (with colour support)
- **3DS** format
- **POV** format
- **DXF** format
- **NFF** format

c) File types supported for export

- **ASCII and binary STL**
- **TRI** format
- **OBJ** format
- **RAW** format
- **VRML 1.0** format



d) Error correction supported

- This software is used as a visualisation tool and does not support any error correction.

e) Salient and added features

- The program includes special rendering options, which can improve the visualisation of the model. The application also supports numerous file formats that are most useful for changing a file into an STL file from other sources.

13.5 Summary

This chapter will be concluded with a brief comparison between the software discussed in this chapter and STLComplete, the prototype propounded in this dissertation.

Table 13.1: Software comparison

	STLview	Magics RP	STeaL	STLComplete
Visual interface	Yes	Yes	Yes	Yes
Error reporting as text file	No	No	No	Yes
Visual error reporting	No	No	No	Yes
Loading and saving of STL files	No	Yes	Yes	Yes
Syntax evaluation	No	No	No	Yes
Duplicated-triangle checking	No	No	No	Yes
Duplicated-triangle correcting	No	No	No	Yes
Vertex-to-vertex rule checking	Yes	Yes	No	Yes
Vertex-to-vertex rule correcting	Yes	Yes	No	Yes
Checking triangle orientation	Yes	Yes	No	Yes
Correcting triangle orientation	Yes	Yes	No	No
Checking Euler's rule for legal solids	No	No	No	Yes

The problems associated with the STL file format have prompted the development of software capable of solving them. Not all errors can be successfully fixed, though, and many of the corrections made will yield a solid with imprecise and unsatisfactory results. In such cases, the file must be created from scratch and tested again to ensure that it is indeed error-free.

Chapter 14

Conclusion



14.1 Introduction

Since the design and implementation of the first LM device, the precision and time required to build a model have improved exponentially with every new model over the past few years. Many new methods for LM have also been propounded and successfully implemented during this period. Although each of these methods has advantages and disadvantages, the disadvantages are slowly being eradicated, thanks to continued research in this realm.

With telemanufacturing, the power of the Internet is truly being harnessed fully. Not only are more companies and individuals benefitting from this technology, but LM machines are also being used optimally, and have since become a source of revenue for many prototyping bureaux.

An important issue to be addressed in the telemanufacturing arena is to find an acceptable standard in terms of which to communicate objects to the bureau. The STL file standard is currently one of the most widely used. Unfortunately, the standard is being hampered by a number of serious flaws, which need to be rectified. The present dissertation constitutes an attempt to investigate these flaws and possibly to find solutions to them.

14.2 Correcting STL files

The most widely used file format in telemanufacturing today is that of the STL file format. Even though the files generated by this format are bulky and fraught with errors, the format is remarkably simple, with the result that it has become the *de facto* standard of telemanufacturing.

Verifying an STL file is commonplace in many telemanufacturing facilities as a precaution before building a model. Since the format is relatively new and many software packages are still in their infancy as far as development is concerned, many models are still being rejected owing to defects and flaws.

Fixing a file, instead of requesting that the client resubmit it, will save valuable time and resources. Oft-times, a triangle is merely incorrectly orientated or has been left out altogether, which would be easy to remedy. In case of some errors, however, the corrected file should be verified with the client to ensure that it is, in fact, the model the user requires.

14.3 Practicability of the STL format

The question still remains, however, whether a format should be used that is all that susceptible to errors. It is the opinion of the author that this particular format should indeed be shelved with all the other outdated formats. Although the simplicity of the format makes it an attractive option for some applications, the overheads it engenders owing to the verification and correcting phases required does not justify its few advantages.

For a truly effective telemanufacturing infrastructure, a file format is required that would meet a certain set of requirements. On closer investigation, it has become evident that the STL format fails to meet the bulk of these requirements.

The criteria for a good file format include the following:

a) Flexibility

The format should be able to store any type of geometrical occurrence with ease, including spherical shapes and curvatures. The STL format adopts a triangle approach and although this is highly effective for describing flat surfaces, curvatures pose a serious problem [3].

b) Geometry intact

The format should also keep the geometry of the object intact. This means that the object should be easily scaled, without any loss of precision. Once a model has been described as a set of triangles (such as by the STL format), it loses all its geometrical information [2].

c) File size

Although complex models do understandably engender large files, these files should not be so large as to hamper the transmission or the storage phase of the file during the telemanufacturing process. As was shown in an earlier chapter, STL files contain huge chunks of redundant information and normally result in cumbersome files for fairly complex models [3].

d) Robustness

The file should also be defined in such a way so as to preclude errors in the model from slipping through. As we know, the STL format is extremely error-prone [2].

e) Simplicity

An important attribute of the file should be its simplicity. It must be easily interpreted by software and should facilitate easy generation of the file from the model. This is the only constraint that the STL format truly meets. The simplicity of the file cannot be overemphasised, but, sadly, this does not compensate for its other shortcomings.

14.4 Future research

Telemanufacturing and Layered Manufacturing are emerging technologies that will benefit many people, from the most powerful engineering company down to the entrepreneur attempting to give shape to a new idea.

Many aspects of the technology still require extensive research, whilst some areas must merely be refined. Some areas of possible future research include the following:

- The possibility of a fully automated telemanufacturing facility. After the file has been uploaded, it could be automatically checked and corrected, scheduled and finally built. Automated hardware that removes the part itself may also be incorporated with the system. A system such as this may

save on human resources by minimising the number of human operators required.

- A software agent that could possibly optimise the orientation of the model by comparing several parameters to user preference. The time to complete may, for example, be deemed less important than the accuracy rating of the part.
- The standardisation of a file format that meets the requirement, as stated earlier in this chapter.
- The implementation of a five-pillar security model (as set out in chapter 5) during the transmission of files, as well as at the telemanufacturing bureau.
- Visualisation of the building process over the Internet, enabling clients to monitor submitted jobs, either directly through a live video feed or through a simulation. The implementation of security in this regard also warrants our further attention before it could be successfully implemented.

Layered Manufacturing and telemanufacturing are still infant technologies, even though they have been around for a few years now. Looking at it philosophically, however, we could argue that man has barely perfected the mechanical clock and that many aspects of life and society could stand improving. This also goes for the computer industry and the Internet and, therefore, for LM and telemanufacturing.

Although at atomic level, Layered Manufacturing, also called “nanometallurgy”, as well as the perfect means of communication, boasting watertight security and lightning-fast speeds, may be music of the future as yet, such should be the stuff of our dreams if we hope to make any progress at all!

14.5 Summary

As long as the STL file format is being used in the Layered Manufacturing and telemanufacturing arena, there would be clamant need to verify files against

errors. Errors in STL files will, however, continue to crop up and will always remain a problem for as long as the format is being used [2].

Although not the ideal scenario, it is the author's opinion that the STL file will remain in use for some time to come, and it is for this reason that various methods should be investigated, applied and improved to verify and finally correct the model.



Appendix A

STLComplete: a user's guide



A.1 Introduction

STLComplete is a utility that allows the user to verify and correct both ASCII and binary STL files before submitting them to a telemanufacturing bureau. It boasts unique features and an intuitive interface, which allow easy visualisation and manipulation of the object to be built.

A.1.1 System requirements

The following minimum requirements for the verification and correction of relatively simple models obtain to the prototype:

- A Pentium 120 MHz processor or a similar processor.
- 16 Mb of system memory.
- A PCI or AGP video card (3D accelerated recommended, but not compulsory).
- 10 Mb of hard-drive space.
- A mouse and a keyboard.

The system should have the Microsoft Windows operating system with OpenGL installed before STLComplete can be executed. Smoother visualisation, as well as faster verification and correction of STL files, is possible with more powerful hardware.

A.2 Moving around

The user can load and save files, export between formats, check and correct errors and translate and rotate the object by using the interface provided.

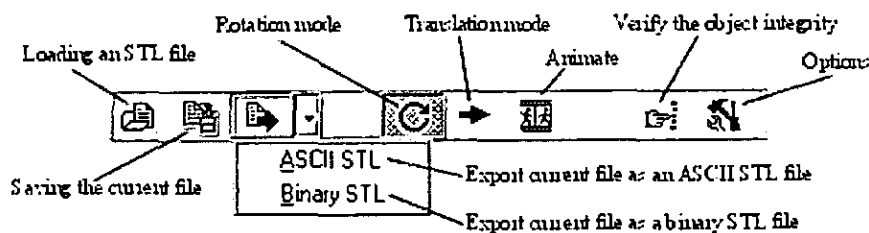


Figure A.1: The toolbar for quick access

The interface consists of a pull-down menu and a toolbar, which make for easy access to the most frequented functions.

a) Loading and saving an STL file

By clicking on the “Loading of an STL file” button located on the toolbar or by selecting the “Load” option under the file menu, the user will be prompted for a file name. In this dialogue box, the user can search the current system and load the file into memory.

After the file has been selected and loaded, the object is displayed in the canvas area, from where the user can move the object around and rotate it in any direction. Operations such as verification and correction are also made available at this stage.

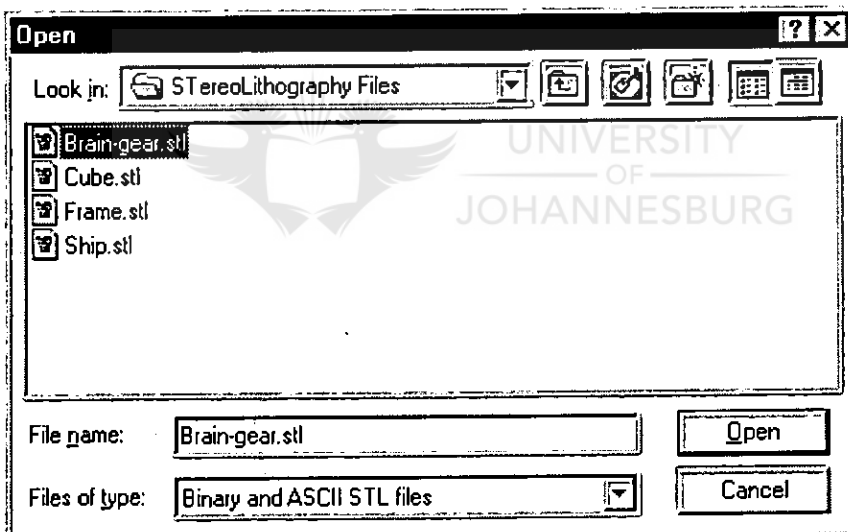


Figure A.2: Open-file dialogue

After a model has been corrected, the user can resave it, so that all the changes can be effectuated. The computer will overwrite the previous file and will retain the original format (ASCII or binary). In order to save the corrected object to a different file or to change the format, the user must make use of the “Export” function.

b) Exporting an STL file

Once successfully loaded, the current object can also be exported to the binary or the ASCII STL format. The solid identifier found in the ASCII variant will be used as a binary header, and vice versa.

c) Rotation and translation mode

Rotation and translation are achieved using the mouse. Either one of the two buttons will be selected for a specific function.

With translation, the user will be able to move the object around by left clicking on it and by then moving the mouse about. Right clicking on the object and moving the mouse vertically will allow scaling of the object.

When rotation is selected, the user will be able to rotate the object by left clicking on it. Vertical mouse movement will achieve rotation around the x axis, while rotation around the y axis is achieved by moving the mouse horizontally. Rotation around the z axis can be achieved by right clicking on the object and by moving the mouse horizontally. Vertical mouse movement while holding the right mouse button down will, once again, scale the object.

The following table will hopefully serve as a handy reference. Please note that the "T" refers to translation, while the "R" refers to rotation.

Table A.1: Rotation and translation table

	Left mouse button	Right mouse button
Mouse UP	T: moving object upward R: x-axis rotation (anti-clockwise)	T: Zoom out R: Zoom out
Mouse DOWN	T: moving object downward R: x-axis rotation (clockwise)	T: Zoom in R: Zoom in
Mouse RIGHT	T: moving object to the right R: y-axis rotation (anti-clockwise)	T: No effect R: z-axis rotation (clockwise)
Mouse LEFT	T: moving object to the left R: y-axis rotation (clockwise)	T: No effect R: z-axis rotation (anti-clockwise)

d) Animation

The user also has the option to rotate the object along the x, y and z axes simultaneously, without using the mouse. By clicking the "Animate" button on the toolbar or by selecting the option from the menu, the computer will automatically rotate the object. This is a useful function for inspection and presentation purposes if the user's attention were divided. Clicking on the button again will stop the animation.

e) Verification of object integrity

After an object has been successfully loaded, the user can verify the syntactical and geometrical integrity of the file. This process can be initiated by selecting the option under the file menu or by clicking on the corresponding toolbar button. The software will now analyse each aspect of the object, while indicating the progress in the status bar at the bottom of the display.

f) Options

By selecting "Options" under the file menu or by clicking on the appropriate toolbar button, the user will be presented with a list of options that takes effect during the error-checking and -correction phase. The options screen is shown in figure A.3 below:

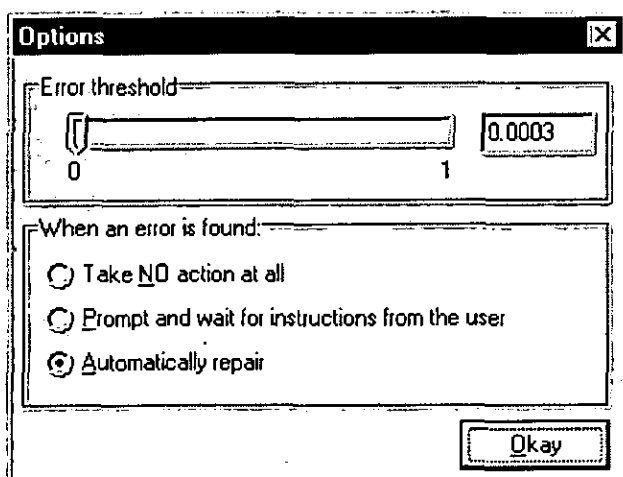


Figure A.3: Program options

The error threshold is useful in cases where vertices that are very close to each other should coincide. Here, the user can select a value ranging from 0.0000 (no vertices will coincide) to 1 (vertices in a radius of 1 unit will coincide).

The next three options will allow the user to specify the action that should be taken once an error has been uncovered:

- **Take NO action at all:** the object is checked for errors and an error report is generated, but the program will make no attempt to correct any of the errors.
- **Prompt and wait for instructions from the user:** if an error were found, the computer would report it and would wait for confirmation before making any correction attempt.
- **Automatically repair:** the computer would immediately attempt to correct each error uncovered.

g) **Object type**

Under the visualisation menu, the user has the option of selecting the rendering method of the model. Three options are presented to the user and each adds a unique benefit during the visualisation of the model. Two of the said rendering methods are shown in figure A.4 below:

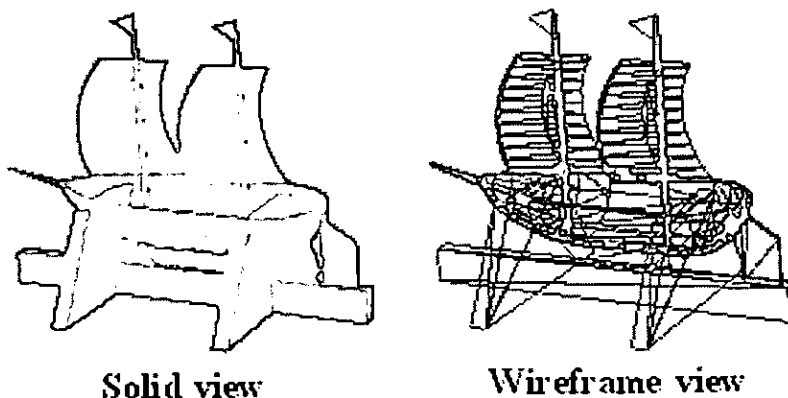


Figure A.4: Solid and wireframe visualisations

- **View as solid:** the most typical visualisation mode. When the model is rendered as a solid, the sides are shaded in such a way as to give the impression of a real object.
- **View as wireframe:** during the rendering of this feature, only the vectors making up the individual triangles are drawn. This is useful when the user merely wants to view the individual triangles.
- **View as points:** now only the vertices are shown, making up the entire object. Features such as rotation, translation and scaling are, however, still available when the object is viewed as a collection of points.

These three options can be found under the visualisation menu, with the methods listed under the “object type” sub-menu. This is illustrated in figure A.5 below:

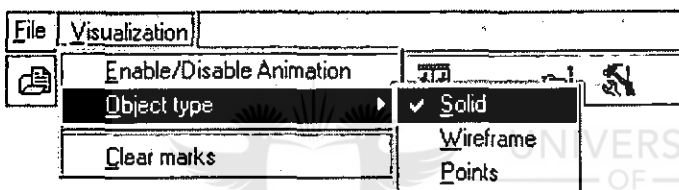


Figure A.5: Selecting the rendering method

h) Clear marks

Once a model is found to be defective, the problem areas are marked in various colours. By selecting this option, the marks on the object will be cleared. This feature can be found under the visualisation menu.

A.3 The verification and correction of errors

Once the object has been successfully loaded, the verification and correction of the model can commence. The first task would be to select the appropriate option under the “error options” menu. (Please refer to section A.3 (f) for details on invoking the options screen.) It is recommended that the error threshold initially be set to zero and that the user be prompted if and when an error be found.

The verification process can be initiated by selecting the verification button on the toolbar or selecting it from the menu. The status bar at the bottom of the screen will indicate the progress, as well as the name and location, of the file being checked. Once an object has been verified, it is rendered again and problematic regions are marked in colour. Each colour indicates a different type of error, as listed in table A.2 below:

Table A.2: Colour legend

Colour	Error present
Red	Duplicated triangles found.
Blue	Triangle with incorrect orientation and/or incorrect normal.
Green	The vertex-to-vertex rule is not complied with.

Depending on the user's selection in the options screen, the program will prompt on each error, automatically correct each error or ignore all errors altogether. More serious errors cannot be corrected, however, and the system will report these to the user. If this were to happen, the file would need to be resubmitted.

At the end of the process, an error report will be generated, which will be saved in the same directory under the same name as the object, but with a .TXT extension. This file will contain basic information of the file in question and a detailed analysis of each error that was verified and, if any such errors were found in the object, whether or not they were successfully rectified.

A.4 Technical specifications

Table A.3: Technical specifications

Programming language	Borland Delphi 5.00 (build 5.62)
Other technologies	OpenGL
Lines of code	6,277 lines
Code size	377,000 bytes
Data size	8,229 bytes
Executable size after compilation	790,528 bytes

Glossary

Algorithm	A step-by-step procedure or method executed or followed by a computer program to accomplish a set task.
American Standard Code for Information Interchange STL (ASCII STL)	An STL file format that is coded using only the standard 7-bit ASCII.
Binary STL	An STL file format using a full 8-bit IBM code for file representation.
Backus Naur Form (BNF)	A set of definitions describing a programming or scripting language.
Computer Aided Design (CAD)	A paradigm in terms of which a design is produced with the aid of a computer system.
Common Gateway Interface (CGI)	Scripting language used on Internet systems.
Compression	A method in terms of which a file is reformatted to take up less storage space.
Compression ratio	The ratio defined as the original file size to the compressed file size. A higher ratio will, therefore, describe a higher level of compression.
Convex polyhedron	A solid object with no enclosed holes or gaps.
<i>De facto</i> standard	A standard that exists by virtue of its widespread use.
Extension	<i>See File extension.</i>
File extension	The last characters following the full stop in a file description.

Hacker	A person who does programming for sheer enjoyment. The term has, however, of late been used in a negative sense to describe a person who attempts to gain access to resources by unlawful means.
Hypertext Markup Language (HTML)	A format used on the World Wide Web to describe the content and layout of a Web page.
An Internet packet	Smallest unit of data sent on the Internet.
Java	A computer language widely used on the Internet, thanks to its compatibility with most systems and browsers.
Normal	See <i>Surface normal</i> .
Operating System (OS)	A system that controls the hardware of a computer system and enables users and applications more readily to interact with the hardware.
Open System Interconnect (OSI) model	A protocol standard developed by the International Standards Organization to implement an open system.
A packet	See <i>Internet packet</i> .
Photopolymer	A liquid substance used in the stereolithography process to create objects.
Protocol	Method or manner of communication.
Prototype	A model created for testing purposes.
Recursive procedure	A routine (function or procedure) calling itself from within itself, which will terminate in a given set of conditions, called the "base case".
Stepping motor	A high-precision electric motor that is capable of accurately turning a defined fraction of a degree. Mainly used in control systems.
Support structures	Structures incorporated in LM to allow the building of arcs and overhangs, which are removed after construction.

Surface normal	A vector of length 1, which is perpendicular to a face or a triangle in STL files. Indicates the outside of the model.
Syntax	Method or set definition of a statement.
Tessellation	The transformation of an object into a set of finite triangles.
Web page	A page found on the World Wide Web (WWW) that could contain text, pictures and links to other pages or resources.



List of sources consulted

- [1] RP&M: Fundamentals of stereolithography,
Paul F. Jacobs, ASME Press, 1992

- [2] Telemanufacturing: Rapid Prototyping on the Internet with automatic
consistency-checking, Michael J. Bailey,
<http://www.sdsc.edu/tmf/Whitepaper/whitepaper.html>

- [3] Design by composition for Rapid Prototyping,
Michael B. Binnard, Stanford University Press, Feb 1999

- [4] SDSC telemanufacturing facility: solid results, tangible benefits,
January-March 1999, Volume 15, Number 1,
<http://www.npaci.edu/envision/v15.1/tmf.html>

- [5] Computational and theoretical problems in modern Rapid Prototyping,
Mark R. Cutkosky, Stanford University Press, 1999

- [6] Building block design for layered shape manufacturing,
Michael B. Binnard and Mark R. Cutkosky,
Stanford University Press, Sept 1998

- [7] Stereolithography and other RP&M technologies,
Paul F. Jacobs, ASME Press, 1996

- [8] Rapid Prototyping from a computer scientist's point of view,
Rapid Prototyping Journal, Volume 2, Number 2, 1996,
A. Dolenc and I. Mäkelä,
MCB University Press, 1996, <http://www.emerald-library.com>

- [9] Telemanufacturing facility, San Diego Super Computer Centre,
<http://www.sdsc.edu/Publications/TMF>
- [10] Telemanufacturing,
Emile Marais, MSc dissertation, RAU Press, March 1998
- [11] Telemanufacturing project with UPM/UQ,
Lawrence Lau and Kevin Burrage, November 1997,
<http://eng.upm.edu.my/home/rasid/html/proposal.htm>
- [12] Understanding data communications and networks,
William A. Shay, PWS Publishing Company, 1995
- [13] Getright, Headlight Software,
<http://www.getright.com>
- [14] Rapid Prototyping in Europe and Japan, Michael J. Wozny,
Chapter 8: Interface formats, March 1997 (WTEC Hyper-Librarian),
<http://itri.loyola.edu/rp/08-05.html>
- [15] Optimised geometry compression for real-time rendering,
Mike Chow, IEEE Visualisation, 1997,
<http://home.earthlink.net/~mmchow/gcompiler/gcompiler.html>
- [16] Graphics file formats FAQ: where to get file format specifications,
James D. Murray,
<http://faqs.bilkent.edu.tr/faqs/graphics/fileformats-faq/part3/section-144.html>
- [17] How to use FTP, TSCNET (C) 1996-97,
<http://www.tscnet.com/faq/hardware/hardwarefaq010.html>

- [18] Information security,
S.H. von Solms and J.H.P. Eloff, RAU Press, 1997
- [19] Client/Server Internet environments,
Amjad Umar, Prentice Hall PTR, 1997
- [20] Telemanufacturing facility: automated STL file-checking (SDSC),
<http://www.sdsc.edu/tmf/Upload/upload.html>
- [21] Telemanufacturing facility: STL format description (SDSC),
<http://www.sdsc.edu/tmf/stl.htm>
- [22] Graphics Gems IV,
P.S. Heckbert, A.P. Professional, 1994
- [23] Sphere generation,
Paul Bourke, May 1992,
<http://www.mhri.edu.au/~pdb/modeling/sphere>
- [24] STL-Pack version 0.33,
Evgeny Ketsuba, 1997-1999,
<http://www.laser.ru/stlpack/>
- [25] Computational methods for Rapid Prototyping of analytic solid models,
Rapid Prototyping Journal, Volume 2, Number 3,
Rida T. Farouki and Thomas König,
MCB University Press, 1996, <http://www.emerald-library.com>
- [26] Winzip Computing,
<http://www.winzip.com>

- [27] ARJ Software Inc, July 1999,
<http://www.arjsoftware.com>
- [28] Optimised geometry compression,
Mike Chow, IEEE Visualisation, 1997,
<http://home.earthlink/~mmchow/vis97/talkp0.0.html>
- [29] A flexible file format for solid freeform fabrication,
Stephen J. Rock and Michael J. Wozny, 1991,
<http://www.rpi.edu/~rocks/PUBS/SFF91-FlexFormat-Abstract.html>
- [30] CAD requirements for Rapid Prototyping tutorial,
John F. Miller, Rapid Prototyping & Manufacturing '94,
Society of Manufacturing Engineers, 1994
- [31] Mathematics junkyard, proof for Euler's rule for legal solids,
David Eppstein, March 2000,
<http://www.ics.uci.edu/~eppstein/junkyard/euler>
- [32a] OpenGL, Silicon Graphics, 1998,
<http://www.opengl.org>
- [32b] DirectX, Microsoft Corporation,
<http://www.microsoft.com/directx>
- [33] Compilers, principles, techniques and tools,
Alfred V. Aho, Ravi Sethi and Jeffrey D. Ullman,
Addison-Wesley Publishing Company, 1986
- [34] Compiler construction,
Niklaus Wirth, Addison-Wesley Publishing Company, 1996

- [35] Rapid Prototyping in Europe and Japan,
Friedrich B. Prinz,
Chapter 8: Model preparation, March 1997 (WTEC Hyper-Librarian),
<http://itri.loyola.edu/rp/08-06.html>
- [36] Calculus, one and several variables,
S.L. Salas and Einar Hille,
John Wiley and Sons, 1990
- [37] Optimal orientation with variable slicing in stereolithography,
Rapid Prototyping Journal, Volume 3, Number 3, 1997,
F. Xu, Y.S. Wong, H.T. Loh, J.Y.H. Fuh and T. Miyazawa,
MCB University Press, <http://www.emerald-library.com>
- [38] Accuracy issues in CAD to RP translations,
George M. Fadel and Chuck Kirschman,
MCB University Press, 1995
- [39] STLview Software,
Igot G. Tebelev,
<http://www.cyberware.com/stlview>
- [40] Materialise, RP Magics and WWRP product information, 1997,
<http://www.materialise.com>
- [41] CIP Software, 1999,
STeaL STL visualisation tool, 1999,
<http://www.sobi.org>