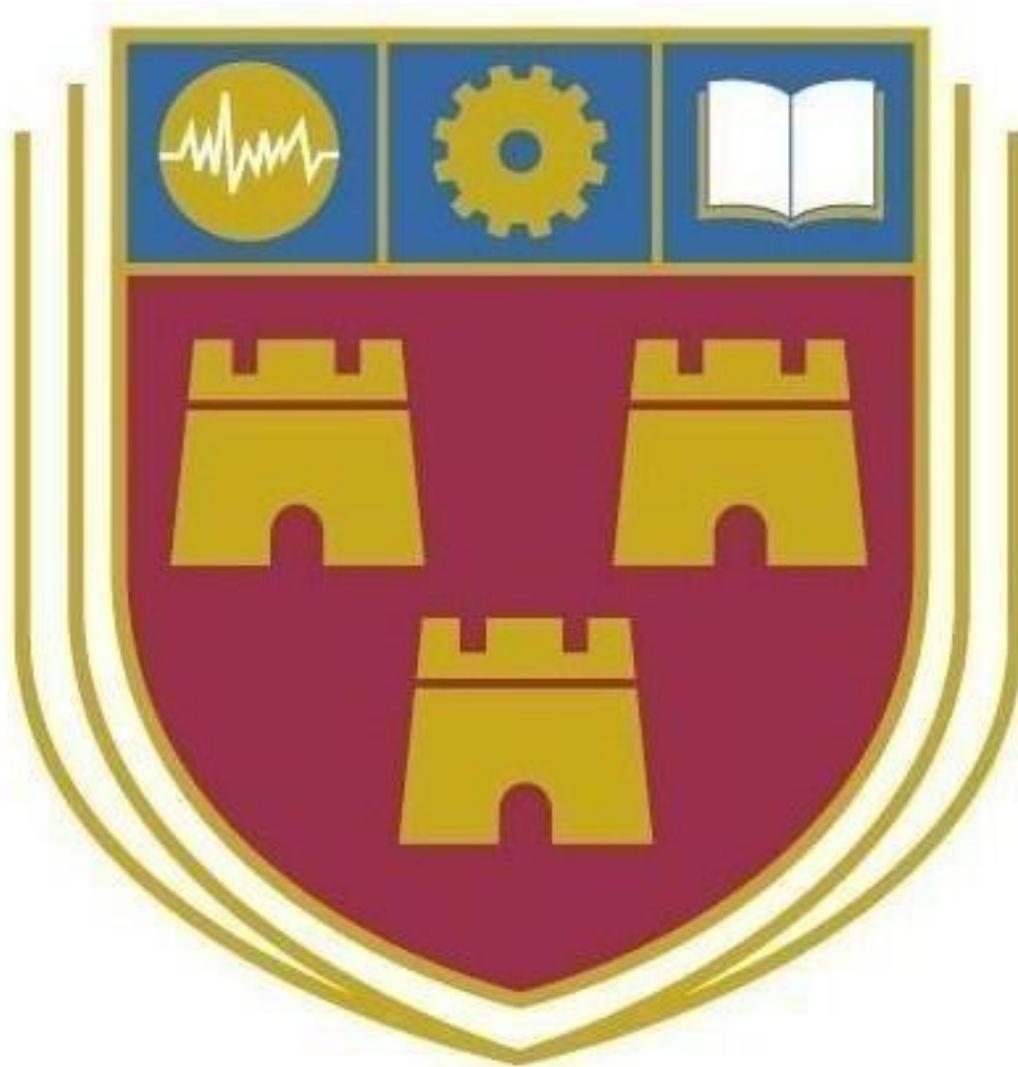


Step File Converter Final Report

Alan Halpin - Institute of Technology Carlow

Started 25/04/2021



Supervisor: Joseph Kehoe

Student Number: C00229361

Abstract

The purpose of this report is to document the problems encountered, what was achieved, what was not achieved, what was learned, differences between earlier design and additional research required.

Table of Contents

Abstract	1
Introduction / Project Description	3
Overview	3
Programming Language	4
C++	4
Development Environment.....	4
Windows	4
Visual Studio 2019.....	4
Version Control Software.....	4
GitHub	4
STEP Format	5
The Algorithm / Problems Encountered	7
STEP File Parsing + Storage of STEP internally	7
Creation of an object to fit the original object	9
Identification of High Level Features	10
Successfully writing STEP files.....	13
Achieved.....	14
Not Achieved.....	14
Stability Calculations.....	14
Challenges	14
Changes from Original Design.....	14
What I would do differently starting again	15
Learning Outcomes	15
Plagiarism Declaration	16
Declaration.....	16

Introduction / Project Description

The STEP File Converter is a tool for CNC engineers, designed to automate a lengthy and costly process. Nothing like this application currently exists. The value is derived from the process being automated. This process involves performing vibration analysis on different permutations of an object until a satisfactory sequence of faces is found. Vibration analysis is performed to ensure the piece worked on remains stable throughout the cutting process. The proposed tool will extract geometrical data from a STEP file and catalogue the different faces of the object described. High level features will be identified from these faces. One by one these features will be subtracted from a solid block shape. Vibration analysis is performed between each step, if the piece does not pass the requirements then a different face will be used. This will eventually create a list of faces to cut in order. Ex. A,B,F,G,C might be an order in which to carve first. Although vibration analysis already exists within various CAD applications, the process described only exists in a manual implementation. This tool will use the Solid Works API to outsource the calculations and identify the optimal cutting order within the application. An engineer will pass a STEP file to the program and the program will start working on the results. Depending on the parts complexity this process may take some time. Typically this process is done manually and can take up to three weeks to complete, this tool hopes to automate that process and reduce that time down to one day.

Overview

The goal of this project was to automate the process of vibration analysis on parts before they are mass produced. This vibration analysis tells engineers which features should be cut first to ensure stability throughout the entire cutting process. This is currently done manually.

Initially it was intended to outsource these complex calculations via an external software API like SolidWorks which has these calculations built into its functionality. Unfortunately this API requires a premium subscription to gain access to. The college offers a student version of solid works which does not have access to this API. To gain access to this API, a large subscription fee would be required (Upwards of 1000 euro). The result of this is that half the functionality requiring API access isn't possible. Other solutions were sought after. Including Fusion 360, this was a dead end as their API did not have the functionality I was looking for.

Programming Language

C++

C is a compiled language developed in 1978. C++ is an extension of C and is a high level object oriented language created in 1998 by Bjarne Stroustrup. C++ differs from C, as C is a process oriented language. C++ can be used for a wide range of applications such as Video Games, GUI Apps, Web Browsers, Advanced Computations and Graphics and backend applications. C++ is applicable to all of these as it is a very fast language and provides a lot of control over hardware. C++ also tackles real world problems to model / calculate abstraction, encapsulation, interface and polymorphism. C++ was chosen for this project as the application will be potentially processing thousands of lines within the STEP file to identify faces and High Level Features of objects. It is also compatible with modelling software API's like SolidWorks and Fusion 360.

The version of C++ used in development of the project was C++17.

Development Environment

Windows

This project was developed in a Windows 10 environment. Windows is a very popular operating system. It was chosen for its compatibility with the STEP file reading software such as SolidWorks and Fusion 360. These applications are exclusively on Windows.

Visual Studio 2019

Visual Studio is an IDE that has many features implemented to help the developer, such as integrated GitHub and a useful debugger.

Version Control Software

GitHub

GitHub was used during the development of this project to keep track of changes, backup the project and potentially roll back the version if there were problems.

GitBash for windows was used to do so.

STEP Format

STandard for Exchange of Product model data (STEP) is a file format used to store 3D modelling data that is considered neutral. The format is considered neutral as it contains a complete description of the product which is independent of any software, OS or computer system. The file uses plain text to portray this information, they chose to do this so it is both machine and human readable. These files are written in EXPRESS, a data modelling language.

ISO 10303-21 is used to store data that represents 3D models in CAD and manufacturing information. These files are written using an ASCII structure aka Plain Text, by nature this makes them easy to read. Having one instance (or step) per line in the format #1, #2, #3 etc... These steps can be located in the DATA section. By nature the STEP file does not contain any information that clearly defines any high level features. It is merely the geometrical information required to build the object. These objects are made in a hierarchy structure. Made from vertex points, lines and other product information.

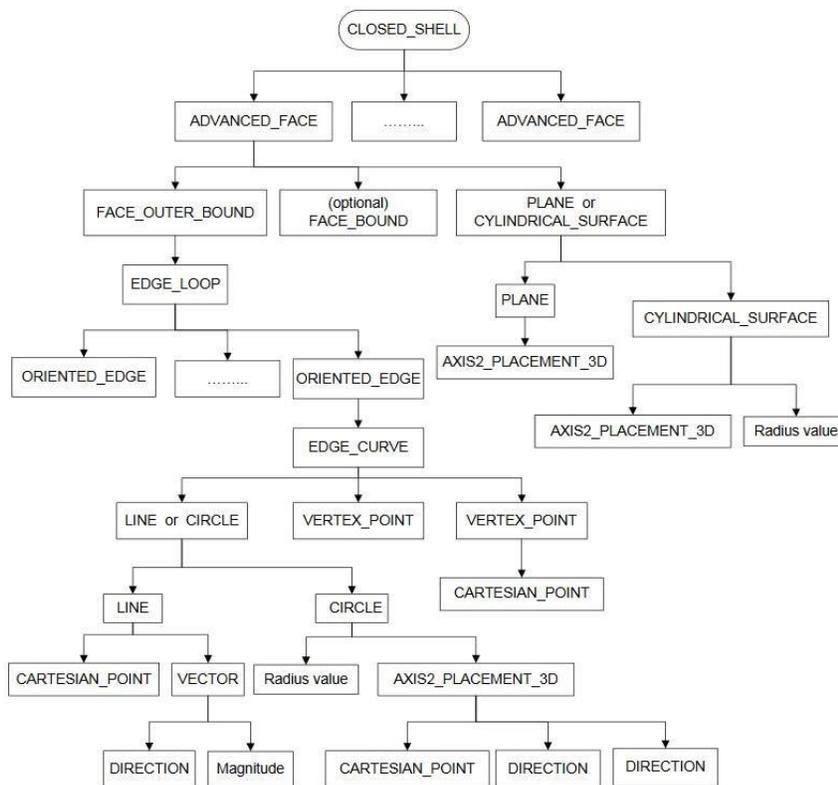


Figure 1: STEP AP203 Data Structure

[Source](#)

The particular version of STEP used during development was STEP AP203.

```

15 DATA;
16 #1 = FACE_OUTER_BOUND ( 'NONE', #269, .T. );
17 #2 = EDGE_CURVE ( 'NONE', #617, #437, #459, .T. );
18 #3 = CARTESIAN_POINT ( 'NONE', ( -50.0000000000000000, 0.0000000000000000, -50.0000000000000000 ) );
19 #4 = CIRCLE ( 'NONE', #655, 10.94457292822576022 );
20 #5 = VECTOR ( 'NONE', #280, 1000.0000000000000000 );
21 #6 = CARTESIAN_POINT ( 'NONE', ( 17.5000000000000000, 100.0000000000000000, 50.0000000000000000 ) );
22 #7 = ADVANCED_FACE ( 'NONE', ( #554 ), #347, .T. );
23 #8 = ORIENTED_EDGE ( 'NONE', *, *, #206, .F. );
24 #9 = CARTESIAN_POINT ( 'NONE', ( -19.9999999999999999, 48.71370998226317539, -22.00985141461442041 ) );
25 #10 = DIRECTION ( 'NONE', ( 0.0000000000000000, -1.0000000000000000, 0.0000000000000000 ) );
26 #11 = VECTOR ( 'NONE', #491, 1000.0000000000000000 );
27 #12 = ORIENTED_EDGE ( 'NONE', *, *, #249, .F. );
28 #13 = CARTESIAN_POINT ( 'NONE', ( -19.9999999999999999, 0.0000000000000000, 15.43548021284641081 ) );
29 #14 = CARTESIAN_POINT ( 'NONE', ( 30.87096042569276122, 62.14829461196278970, 50.0000000000000000 ) );
30 #15 = ORIENTED_EDGE ( 'NONE', *, *, #515, .T. );
31 #16 = CARTESIAN_POINT ( 'NONE', ( -19.9999999999999999, 0.0000000000000000, -22.00985141461425343 ) );
32 #17 = CARTESIAN_POINT ( 'NONE', ( 50.0000000000000000, 50.0000000000000000, -57.0000000000006395 ) );
33 #18 = VERTEX_POINT ( 'NONE', #197 );
34 #19 = VERTEX_POINT ( 'NONE', #88 );
35 #20 = DIRECTION ( 'NONE', ( -1.0000000000000000, -0.0000000000000000, -0.0000000000000000 ) );
36 #21 = CARTESIAN_POINT ( 'NONE', ( -19.9999999999999999, 0.0000000000000000, 15.43548021284641081 ) );
37 #22 = ORIENTED_EDGE ( 'NONE', *, *, #110, .T. );
38 #23 = AXIS2_PLACEMENT_3D ( 'NONE', #293, #47, #408 );
39 #24 = ORIENTED_EDGE ( 'NONE', *, *, #482, .F. );
40 #25 = DIRECTION ( 'NONE', ( -1.0000000000000000, -0.0000000000000000, -0.0000000000000000 ) );
41 #26 = EDGE_CURVE ( 'NONE', #77, #183, #237, .T. );
42 #27 = ORIENTED_EDGE ( 'NONE', *, *, #637, .F. );
43 #28 = DIRECTION ( 'NONE', ( -0.7896391215989457812, -0.6135715586958422341, -0.0000000000000000 ) );
44 #29 = LINE ( 'NONE', #598, #423 );
45 #30 = LINE ( 'NONE', #599, #54 );
46 #31 = EDGE_CURVE ( 'NONE', #432, #428, #29, .T. );
47 #32 = VERTEX_POINT ( 'NONE', #97 );
48 #33 = CARTESIAN_POINT ( 'NONE', ( -50.0000000000000000, 0.0000000000000000, 50.0000000000000000 ) );
49 #34 = ORIENTED_EDGE ( 'NONE', *, *, #206, .T. );
50 #35 = CALENDAR_DATE ( 2021, 25, 4 );
51 #36 = ORIENTED_EDGE ( 'NONE', *, *, #522, .F. );
52 #37 = ADVANCED_FACE ( 'NONE', ( #253, #400 ), #647, .F. );
53 #38 = CIRCLE ( 'NONE', #436, 10.94457292822576022 );
54 #39 = CARTESIAN_POINT ( 'NONE', ( 17.5000000000000000, 65.0000000000000000, 60.0000000000000000 ) );
55 #40 = AXIS2_PLACEMENT_3D ( 'NONE', #444, #238, #175 );
56 #41 = LINE ( 'NONE', #509, #475 );
57 #42 = COORDINATED_UNIVERSAL_TIME_OFFSET ( 0, 0, .AHEAD. );
58 #43 = CARTESIAN_POINT ( 'NONE', ( -50.0000000000000000, 100.0000000000000000, 50.0000000000000000 ) );
59 #44 = DIRECTION ( 'NONE', ( 0.0000000000000000, -1.0000000000000000, 0.0000000000000000 ) );
60 #45 = DATE_AND_TIME ( #362, #586 );
61 #46 = ORIENTED_EDGE ( 'NONE', *, *, #218, .T. );
62 #47 = DIRECTION ( 'NONE', ( -0.0000000000000000, -0.0000000000000000, 1.0000000000000000 ) );
63 #48 = CARTESIAN_POINT ( 'NONE', ( 25.77903920696300588, 23.98704954336139394, -50.0000000000000000 ) );
64 #49 = PERSON_AND_ORGANIZATION_ROLE ( 'creator' );
65 #50 = ORIENTED_EDGE ( 'NONE', *, *, #95, .T. );
66 #51 = CARTESIAN_POINT ( 'NONE', ( -19.9999999999999999, 0.0000000000000000, 0.0000000000000000 ) );

```

Fig 2: STEP File

All of the features can be identified in the DATA section of a STEP file, a high level feature is made up of one or more advanced faces. An advanced face is just a single face made up of edge curves, these edge curves have 2 vertex points and a line. The line contains a cartesian point that references one of the vertex points. It also contains a vector which informs the reader what direction this line is moving.

There is no order to a STEP file, each line may reference forward or backward in the file. There is also no information given to identify which faces belong where other than their X , Y , Z coordinates. The STEP file is all low level information. High Level Features cannot be identified as the information is presented by default.

When first tackling this problem of identifying the high level features I tried creating a function that would use these edge curves to try identify different shapes. So a cuboid would be identified as having 12 lines, 4 lines moving parallel one direction, 4 lines parallel another, and 4 lines parallel another. These lines would also be perpendicular to each other and some intersecting the others. This proved too difficult a task and the implementation I conceived would need to have every

possible shape encountered, manually programmed into the application. This would get too complex as there are near unlimited amounts of high level features possible.

The Algorithm / Problems Encountered

STEP File Parsing + Storage of STEP internally

There are 3 important parts of a STEP file. First there is the header section. This section is just a description of the file. It declares things like the name of the file, the date it was created and what protocol it uses. Then there is the data section, there are two parts to the data section and they are mixed up together. There is the actual data, which is everything from faces, lines and points. Then there is what I refer to as compile lines, these lines aren't related to the object itself in that no part of the object references them. Without these lines I could not read the STEP file using a STEP reader like SolidWorks. These lines contain product information such as what the piece is made out of etc.

The header needs to be read and saved for later and the data list as a whole. In every step the first character is a '#' symbol. There are exceptions to this when a step is multiple lines. The end of a step is denoted by ';'. Once the datalist was made, the next step was to gather up each face along with that face's sub features. #7 = `ADVANCED_FACE ('NONE', (#554), #347, .T.) ;`

This is an example of a typical face. The step references other steps and those steps reference more steps. This creates a tree-like structure. The datalist is searched for an instance of "ADVANCED_FACE", when an advanced face is found, a substring of the line is created to avoid reading the current steps number. The string is searched character by character for other steps. These steps are added to a list of next lines to be searched. This is done repeatedly until there are no more next lines. I used a C++ map to contain this information. The key was the step number of the face and the information stored was a vector made up of every sub-feature of that face including the face itself.

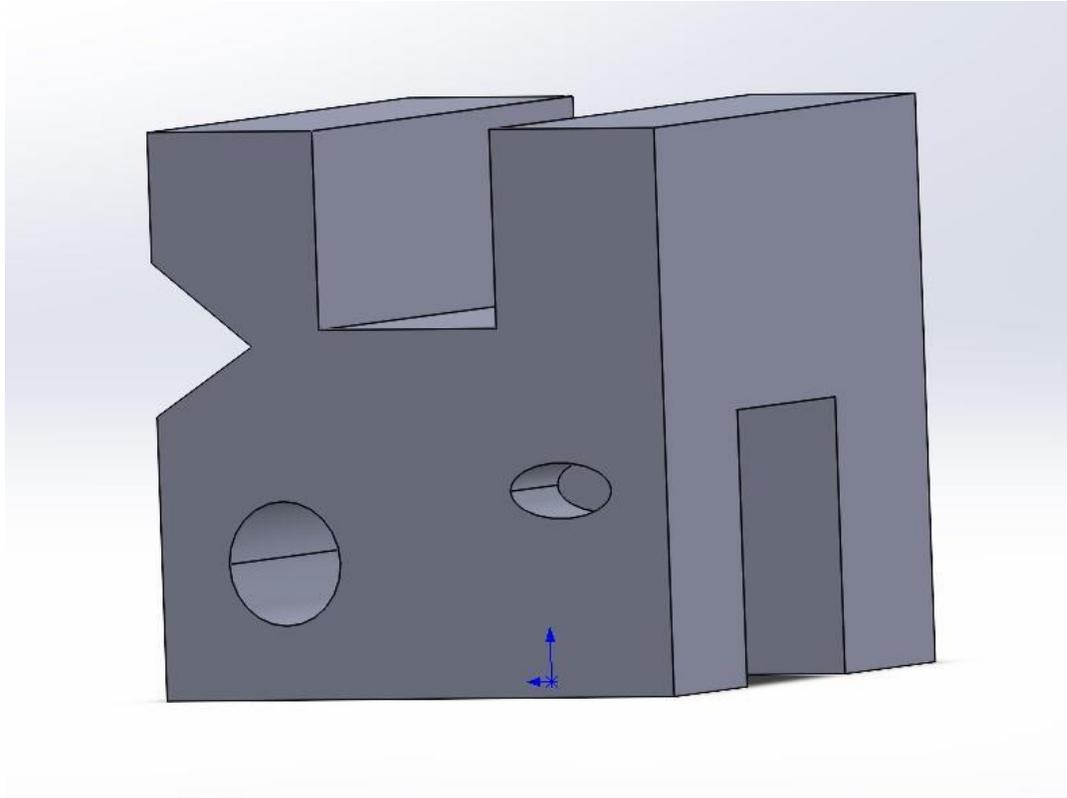


Fig 3: CubeCUTS Object

```
STEP File Location: STEPFILE-Project\STEPFILES\  
"STEPFILES/Cube.STEP"  
"STEPFILES/CubeCircleCut.STEP"  
"STEPFILES/CubeCUTS.STEP"  
"STEPFILES/CubeSlot.STEP"  
"STEPFILES/CubeSQcut.STEP"  
"STEPFILES/CubeSQtopCut.STEP"  
"STEPFILES/CubeVcut.STEP"  
Enter the name of the STEP file (sans file extension)  
CubeCUTS  
File name: CubeCUTS  
Extracting Features...  
STEP File Opened  
STEP File Closed  
  
Advanced Faces Found: 22
```

Fig 4: Parsing Output

Creation of an object to fit the original object

To identify the high level features within an object, one method is to create a new object that is only as big as the min / max coordinates of the original object and then compare the two objects to see which points are shared and which are not.

The original STEP file has been stored internally. The vertex points of that object have been found and stored also. These vertex points are actual points on the object. Within the step file vertex points, point to cartesian points. There are many cartesian points in a STEP file and some of them are not 'real' points. They aren't located or depicted on the object. They are reference points. If there is a curve on one of the faces, there may not be a full circle but the centre point of that arc used to create the curve may not be located anywhere near the physical object itself. So it is important to only use vertex points and not every cartesian point when finding the min / max points of the objects.

```
#21 = CARTESIAN_POINT ( 'NONE', ( -19.999999999998934, 0.000000000000000000, 15.43548021284641081 ) );
```

This operation was done by stepping through each character of the line and searching for characters that are digits. This way, the X, Y and Z can be identified.

That operation is repeated on a previously made cube, used as a blueprint. The next step involves identifying if the vertex points of the cube are the minimum or maximum value of x, y, z. They are replaced accordingly with the min / max values retrieved from the original object.

This image portrays the original object laid over the new object created to fit the min, max. We can visually see where the cuts need to be made.

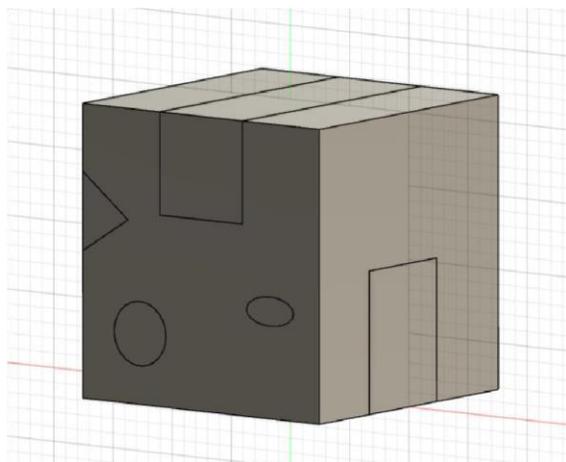


Fig 5: Original Object Displayed with New Object

Identification of High Level Features

As mentioned before, high level features are made from low level features. The STEP file only contains low level features and there is no indication other than x,y,z location what face belongs where. Using the cube created with the min , max points, we compare the two objects. We see which points are not shared. This isn't straightforward however, as sometimes in STEP, the lines that are created are drawn from a slightly different position than the actual point it is drawn from. It is typically a fraction off.

While comparing all points in one object with the other object, checking if they are equal does not work. A function needed to be made to check if they were equal within so many decimal places.

When a point is not found it is saved to a list as not found, next these points need to be located. Which face do they belong to? A face can only be considered not found if all vertex points of the face are within this new list. This is because many of the faces will share 1 point or more. After this search is done we have a list of faces where all the points cannot be found.

Next the high level features need to be identified from these faces. This is done by checking what faces share points with each other. Once this list is compiled we can check if these faces have a face in common, and if those faces that are in common are also in the list of faces where all their points cannot be found. If this is determined to be true then these faces are added to the first high level feature and this is repeated until there are no more faces to choose from.

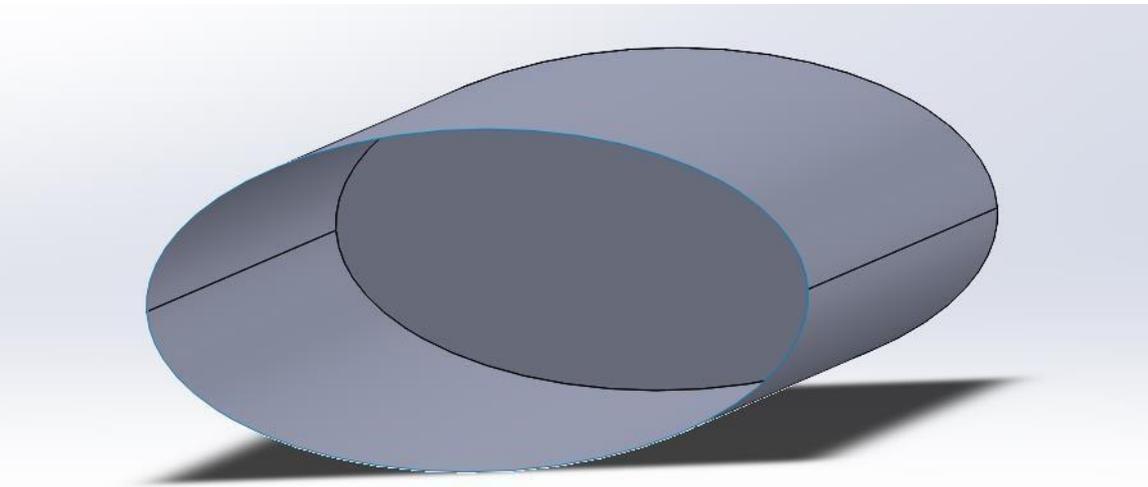


Fig 6: High Level Feature 1

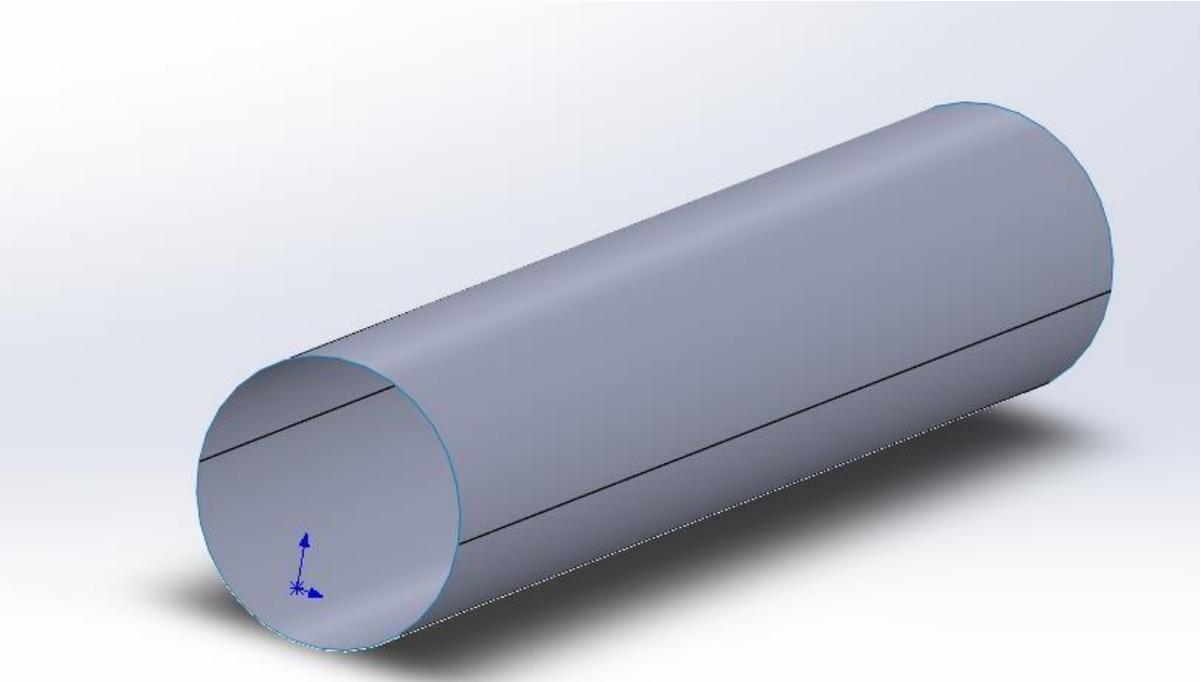


Fig 7: High Level Feature 2

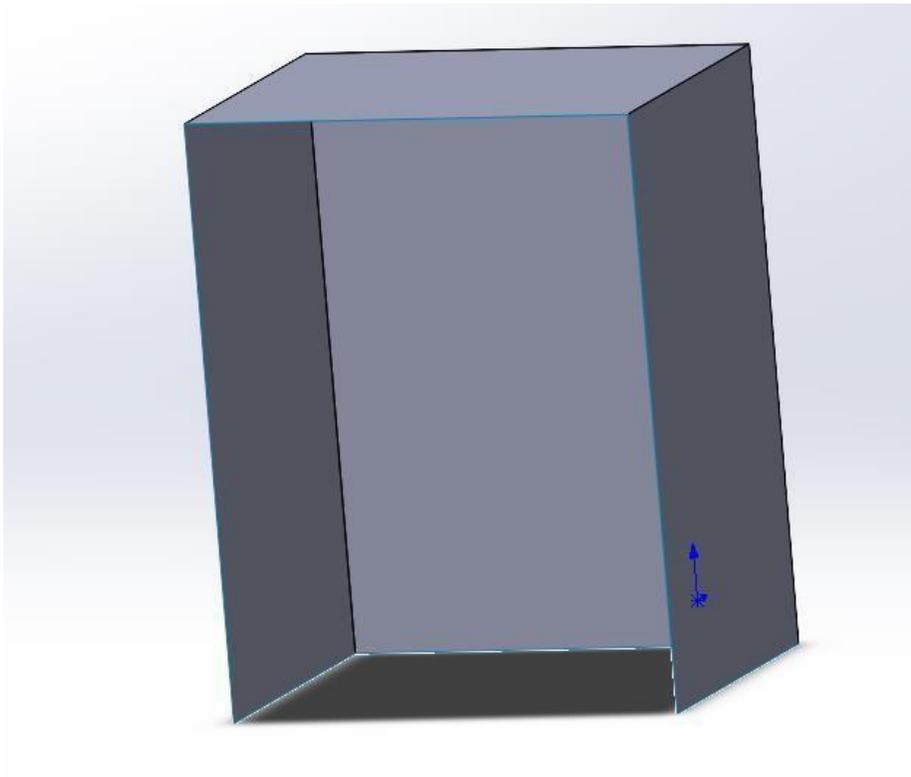


Fig 8: High Level Feature 3

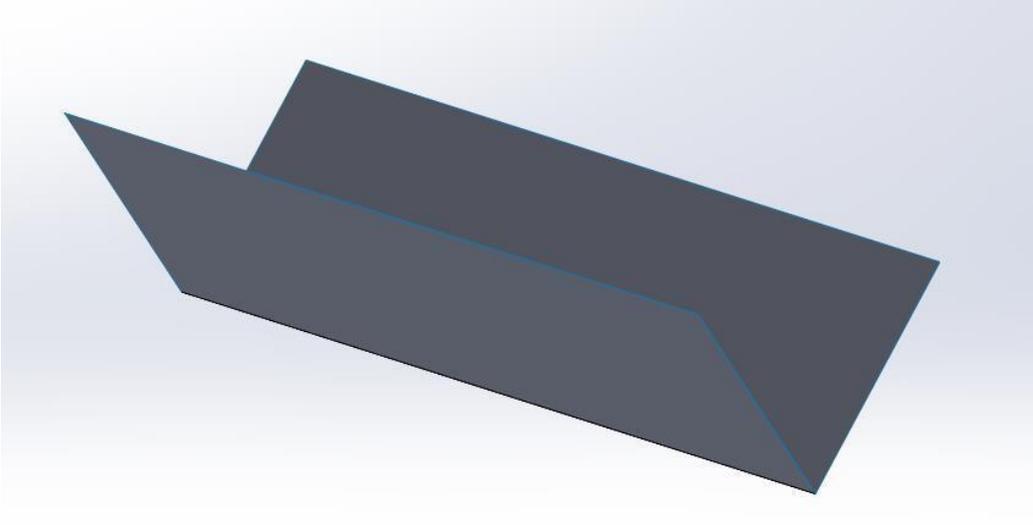


Fig 9: High Level feature 4

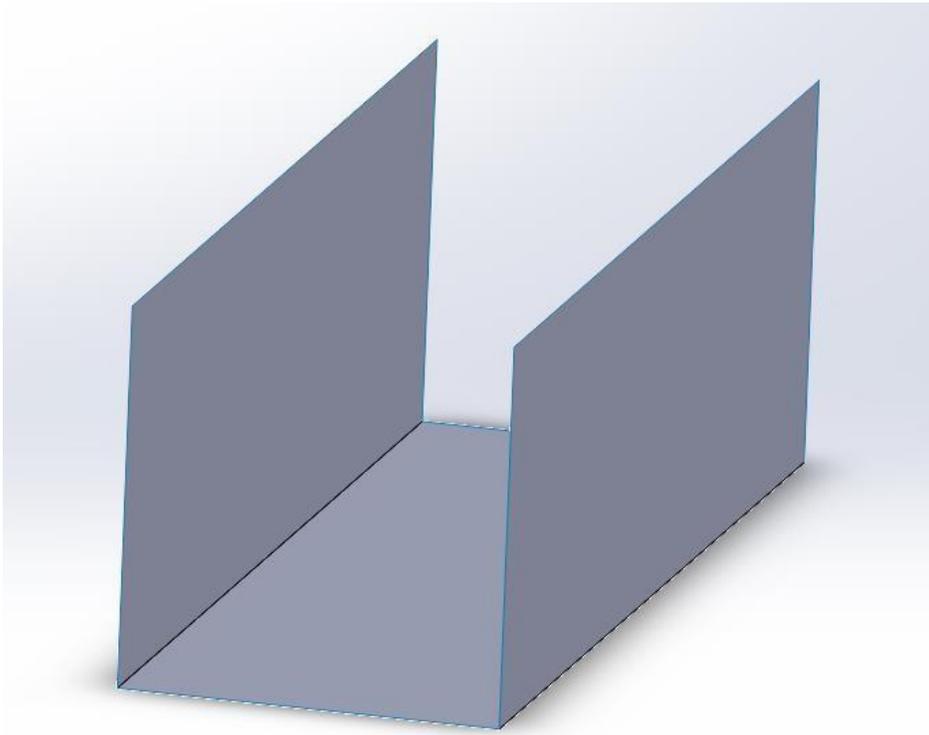


Fig 10: High Level Feature 5

The following high level features have been identified. Cuts need to be made at the vertex points displayed below.

```
High Level Feature : 0
#169 = CARTESIAN_POINT ( 'NONE', ( -19.22074655601591076, 36.36719323738191179, -50.0000000000000000 ) );
#344 = CARTESIAN_POINT ( 'NONE', ( -39.01447881776221038, 36.36719323738191179, -32.0000000000000000 ) );
#381 = CARTESIAN_POINT ( 'NONE', ( -39.01447881776221038, 36.36719323738191179, -50.0000000000000000 ) );
#756 = CARTESIAN_POINT ( 'NONE', ( -19.22074655601591076, 36.36719323738191179, -32.0000000000000000 ) );
Press any key to continue . . .
High Level Feature : 1
#502 = CARTESIAN_POINT ( 'NONE', ( 36.72361213518874479, 23.98704954336140105, -50.0000000000000000 ) );
#693 = CARTESIAN_POINT ( 'NONE', ( 14.83446627873724033, 23.98704954336140105, 50.0000000000000000 ) );
#710 = CARTESIAN_POINT ( 'NONE', ( 14.83446627873724033, 23.98704954336140105, -50.0000000000000000 ) );
#88 = CARTESIAN_POINT ( 'NONE', ( 36.72361213518874479, 23.98704954336140105, 50.0000000000000000 ) );
Press any key to continue . . .
High Level Feature : 2
#104 = CARTESIAN_POINT ( 'NONE', ( -50.0000000000000000, 48.71370998226320381, -22.00985141461439909 ) );
#235 = CARTESIAN_POINT ( 'NONE', ( -50.0000000000000000, 0.0000000000000000, 15.43548021284640015 ) );
#407 = CARTESIAN_POINT ( 'NONE', ( -20.0000000000000000, 48.71370998226320381, -22.00985141461439909 ) );
#480 = CARTESIAN_POINT ( 'NONE', ( -20.0000000000000000, 0.0000000000000000, -22.00985141461424988 ) );
#554 = CARTESIAN_POINT ( 'NONE', ( -20.0000000000000000, 48.71370998226320381, 15.43548021284640015 ) );
#577 = CARTESIAN_POINT ( 'NONE', ( -20.0000000000000000, 0.0000000000000000, 15.43548021284640015 ) );
#600 = CARTESIAN_POINT ( 'NONE', ( -50.0000000000000000, 0.0000000000000000, -22.00985141461424988 ) );
#625 = CARTESIAN_POINT ( 'NONE', ( -50.0000000000000000, 48.71370998226320381, 15.43548021284640015 ) );
Press any key to continue . . .
High Level Feature : 3
#216 = CARTESIAN_POINT ( 'NONE', ( 50.0000000000000000, 50.0000000000000000, 50.0000000000000000 ) );
#400 = CARTESIAN_POINT ( 'NONE', ( 50.0000000000000000, 77.01209037248150935, 50.0000000000000000 ) );
#435 = CARTESIAN_POINT ( 'NONE', ( 30.87096042569275056, 62.14829461196279681, 50.0000000000000000 ) );
#519 = CARTESIAN_POINT ( 'NONE', ( 50.0000000000000000, 50.0000000000000000, -50.0000000000000000 ) );
#580 = CARTESIAN_POINT ( 'NONE', ( 30.87096042569275056, 62.14829461196279681, -50.0000000000000000 ) );
#597 = CARTESIAN_POINT ( 'NONE', ( 50.0000000000000000, 77.01209037248150935, -50.0000000000000000 ) );
Press any key to continue . . .
High Level Feature : 4
#279 = CARTESIAN_POINT ( 'NONE', ( 17.5000000000000000, 100.0000000000000000, 50.0000000000000000 ) );
#367 = CARTESIAN_POINT ( 'NONE', ( -17.5000000000000000, 65.0000000000000000, -50.0000000000000000 ) );
#40 = CARTESIAN_POINT ( 'NONE', ( -17.5000000000000000, 65.0000000000000000, 50.0000000000000000 ) );
#477 = CARTESIAN_POINT ( 'NONE', ( -17.5000000000000000, 100.0000000000000000, 50.0000000000000000 ) );
#589 = CARTESIAN_POINT ( 'NONE', ( 17.5000000000000000, 65.0000000000000000, 50.0000000000000000 ) );
#628 = CARTESIAN_POINT ( 'NONE', ( -17.5000000000000000, 100.0000000000000000, -50.0000000000000000 ) );
#7 = CARTESIAN_POINT ( 'NONE', ( 17.5000000000000000, 65.0000000000000000, -50.0000000000000000 ) );
#748 = CARTESIAN_POINT ( 'NONE', ( 17.5000000000000000, 100.0000000000000000, -50.0000000000000000 ) );
Press any key to continue . . .
```

Fig 11: Program Output

Successfully writing STEP files

Once the list of faces was created, now the ‘compile lines’ as mentioned before could be found. I compared the list of the features with the datalist to find items not present in any face. This part was crucial in order to produce my own STEP files. Otherwise they would not compile. The header also needed to be added along with “DATA;” to denote the data area of the file.

Achieved

To sum up;

This project achieved STEP file parsing, storage of STEP files internally, identification of high level features and successfully writing STEP files.

Not Achieved

Stability Calculations

Most of the specification relied on the stability calculations but unfortunately as I could not gain access to the Solid Works API this feature was not implemented. This part of the application is relatively trivial however. Writing the interface for solid works and implementing the API would be the easy part. Identifying high level features from STEP files was the majority of the workload. Now all that is left to implement is the permutation generator and a means of adding features one by one to an object to be passed to Solid Works. Given the correct licensing, this project would need an extra month to fully implement the API and its vibration analysis calculations.

Challenges

- Read File: Able to read in the Step file format and store internally; Find High level "faces": Produce from low level details the higher level shapes being carved out;
- Apply Stability Equations: Given a shape determine its stability;
- Carving Order Determination: System determines best (most stable) order to carve faces;

Not all of these challenges were met, however that was out of my control. The difficulty of this project was not known from the beginning and only became clear once the project had entered the coding stage.

Changes from Original Design

Other than the missing solid works interface, the general design of the application has stayed as planned. There was a minor design change within the code, I separated the extracting of data and extracting of high level features into separate classes. This made it easier to work with and easier to implement the missing features in the future. Open for extension, closed to modification.

What I would do differently starting again

I would start off with the right language from the beginning to avoid switching and learning a new language half way through. I would also attempt to implement a more coordinate focused means of identifying the high level features. I do not have enough graphical experience to implement something like that in a short amount of time. With more 3D graphics research at the beginning of the project maybe I could have implemented something like that. I did not know what direction this project would take at the beginning. The difficulty and nature of the project was unknown.

Learning Outcomes

At the beginning of this project I had very little experience using C++ and STEP files were completely foreign to me.

I found C++ to be very difficult to learn and work with, C++ is known within industry to be a hard language to grasp. If you learn C++, learning other languages seems trivial. When tackling the problems in this project, I had no reference to help me online as this problem has not been solved before. This project involved a lot of experimentation with both the solving of problems and understanding the language.

I had originally wrote this application in C# but after switching between the different APIs I attempted to use, I had rewritten the program in C++. This added unnecessary pressure as I now had to learn a new language in the middle of the project. I didn't let this ruin the project all together however. I tried my best to learn in a speedy manner.

This project showed me that if I sit down and focus on a problem I can solve it, although it was a difficult process. I have no experience working with 3D geometry and I found it very confusing at times.

There is helpful information on the inner workings of STEP available online but I found, to truly gain an understanding, I needed to experiment and test what happens when I perform certain actions on the data or change values within the STEP file.

This project did not go as planned and turned out to be much more difficult than I had initially thought. However I feel as though I have benefitted from it anyway. I have gained new skills in problem solving, knowledge of a previously unknown language and the inner workings of STEP files.

I hope to bring this project to completion in the near future.

Plagiarism Declaration

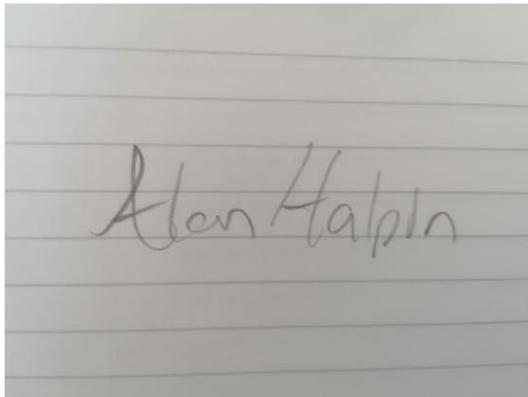
Declaration

- I declare that all material in this submission e.g.thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
- I have provided a complete bibliography of all works and sources used in the preparation of this submission.
- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offense.

Student Name: Alan Halpin

Student Number: C00229361

Signature:

A photograph of a handwritten signature 'Alan Halpin' on a piece of lined paper. The signature is written in dark ink and is centered on the page.

Date: 30/04/2021