CADConverter

For

Converting Complex CAD files in HDF5 Format

by

Sachin Chaudhary
B.TECH., Guru Nanak Dev University, 2016

A Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

CADConverter

For

Converting Complex CAD files in HDF5 Format

by

Sachin Chaudhary
B.TECH., Guru Nanak Dev University, 2016

Supervisory Committee

Dr. Teseo Schneider, Supervisor
(Department of Computer Science)

Dr. Sajin Koroth, Departmental Member
(Department of Computer Science)

ABSTRACT

In this paper, we offer a very effective approach for extracting data from Computer-Aided Design And Manufacturing (CAD/CAM) step files and converting them to the self-descriptive and open-source HDF5 format. This format provides for the seamless integration of data and metadata inside a single file, making it ideal for dealing with complex data. The difficulties associated with converting CAD files between applications and the sophisticated data organisation inside multiple CAD file formats have motivated our research. The international standard ISO 10303 or STEP ('STandard for the Exchange of Product Model data') addresses the format complexity and CAD data conversion problem between different computer-aided design (CAD) systems. We reviewed current CAD datasets like ABC and Fusion360 for geometrical data processing and created an algorithm that converts CAD models to open-source a human-readable format to feed deep learning algorithms directly and hence eliminating the requirement for third-party software for file conversions will be boosting the efficiency of CAD data administration and processing.

To do this, we employ a method that combines data pre-processing as well as the development of an algorithm that converts around one million CAD models to the HDF5 format and uses the derived data for a number of applications, including machine learning and data analysis. The findings of the study lead to a better understanding of CAD data processing procedures and establish the framework for future research and development and integrate the converted dataset with PyTorch and TensorFlow to address current CAD system constraints, such as the limitations of standard neutral 3D CAD file formats that are difficult to understand. It opens the way for more efficient and simplified procedures in CAD-intensive sectors.

Contents

List of Figures

ACKNOWLEDGMENTS

First and foremost, I want to convey my heartfelt thanks to Dr Teseo Schneider for his supervision, enthusiasm, determination, and support during my project. I am really grateful to him for his continual guidance and advice during this endeavour. The University of Victoria and the Department of Computer Science have played an important role in my development by providing an outstanding environment in which I may enhance and perfect my technical and employability abilities.

I must recognise and appreciate my teammate Sai Chandra's amazing efforts and vital contributions to this project. Working with such a hard-working guy has been a genuine honour and fortune for me.I would also like to express my sincere gratitude to Nafiseh Izadyar who contributed to the completion of this project.

My parents, for encouraging and inspiring me all through my journey. It would be impossible to realize this dream without their immense support and love.

My sisters, Dr Anchal Chaudhary and Tanveer Chaudhary, their faith in my skills has been a continual source of encouragement for me throughout my research.

I am eternally grateful to my close brothers Dharminder Chaudhary, Inderbir Singh, Simran Sodhi and Aman Sharma, for their constant encouragement and support. Their presence and understanding during challenging times have made this journey more fulfilling.

Chapter One

Introduction

1.1 Background and Motivation

Computer-aided design (CAD) tools have transformed product design and development in industries such as manufacturing, automotive, and aerospace. Their invention and broad use have had a significant influence on various industries, simplifying procedures and allowing novel design methods [1]. There are a couple of CAD softwares available with their unique file formats such as AUTOCAD including DWG (AutoCAD Drawing) and ONSHAPE (Onshape Document), STEP, Autodesk Inventor, BricsCAD (DWG and DXF). These Different suppliers' CAD software may employ proprietary file formats, causing compatibility concerns when transferring data across systems [2]. This fragmentation makes converting one CAD file version to another challenging, as each requires a third-party software dependency to work. The availability of diverse and enormous data sets opened the door to a broad range of applications, including data analytics, machine learning, and artificial intelligence. In comparison to other fields, there are fewer publicly available datasets, particularly for Computer-Aided Design, and most of them do not contain geometry but simply meshes. To address these problems, several efforts have been made to construct a conversion from CAD dataset to B-Rep Model [3], however, the difficulty of preserving consistency between the modelling levels impedes the CAD model conversion and needs a better approach.

1.2 Contribution and Significance of the Study

In view of the aforementioned issues, we have worked on this project to propose and develop an algorithmic solution for rapidly extracting CAD data and successfully changing it into an open-source HDF5 format. The primary goal of this project is to improve the accessibility and adaptability of CAD information across many sectors and applications by providing an alternate way to CAD step [4] data processing and converting it into a less complicated format. By using an open-source format for CAD models, our initiative aims to eliminate the need for third-party software for file conversions, reducing compatibility difficulties and significantly improving the efficiency of CAD data administration and processing. Notably, the technique developed herein is intended to scale effectively, making it especially ideal for handling large datasets such as the Fusion360 dataset [5] and ABC dataset which is a collection of 1 million CAD models as previously developed in research by Koch et al [6] This conversion feature from step files to open source cad files makes our method a significant tool for data science benchmarking and analysis. Many fields of computer science can be transformed by the combination of massive open-source data sets resulting from this conversion algorithm and deep learning techniques as previously demonstrated in studies by Wu Z [7].

1.3 Organization of the Report

The rest of the project report is organized as follows:

- Chapter Two offers a literature review and an overview of existing methods and approaches related to CAD data conversion.
- Chapter Three introduces the terminology, definitions, and technical background required for understanding the proposed algorithm and its implementation.

- Chapter Four details the system data of the proposed algorithm that supports the conversion and extraction of information from CAD files in the open-source HDF5 format.

- Chapter Five discusses the execution of the proposed algorithm, including the programming language and tools used.

- Chapter Six presents the testing, validation, refinement , and the results obtained from its application to CAD data.

- Chapter Seven summarizes the contributions and findings of the study and its limitations and outlines potential future work and research directions in the field of CAD data processing and conversion.

Chapter Two

Background Work

2.1 Literature Review

2.1.1 CAD Data Interoperability

The use of machine learning approaches has resulted in considerable advancements in the efficiency and accuracy of processing 3D geometric data in recent years, opening up new opportunities for applications in computer graphics, computer-aided design, medical imaging, and many other disciplines. The discrepancies in internal mathematical representation techniques and the internal quality of geometric specifications in the modelling kernels of various CAD systems are the most critical challenges in data transfer. A. Soloviev and M. Kogalovsk[1] address the problem of compatibility among CAD systems in their study work. Salvatore Gerbino [8] notes challenges during the pre-and post-processing phases of CAD data when using a neutral format such as IGES or STEP in his study. Stephen Cawood [9]] investigates the necessity to build a model to enable CAD interoperability and prevent data interchange failure using neutral standard formats. Andreas Egger [10] presents a data model that combines the capabilities of the Standard for the Exchange of Product Model Data (STEP) and the Hierarchical Data Format version 5 (HDF5) in his research paper. It also explores the possible uses and difficulties of implementing the proposed HDF5 data model in several engineering areas.

2.1.2 Data Compatibility

In today's production environment, CAD models are typically given by several suppliers, resulting in a heterogeneous application environment. Despite several integration methodologies established over the last few decades, software integration and product data sharing remain difficult concerns that must be solved. Xi Vincent Wang [11] discusses the lack of an open-source format that allows for easy cooperation across various CAD stakeholders. Andreas Egger [10] investigates the use of STEP (Standard for the Exchange of Product Model Data) and HDF5 (Hierarchical Data Format 5) formats to develop a unified framework for transferring and storing CAD and analytical data..

2.1.3 Surface Reconstruction

Occupancy Networks, which use continuous volumetric functions to describe 3D surfaces, were presented by Mescheder et al [12]. This methodology highlights the promise of neural network-based approaches for recreating 3D surfaces from geometric data by using techniques such as learning implicit representations or occupancy functions. M.Ovsjanikov [13] used Deep Geometric Functional Maps to uncover correspondences and integrate many scans into a single model in the domain of surface registration. Their method blends classic approaches like Iterative Closest Point (ICP) with geometric deep learning to handle non-rigid deformations and large-scale point clouds.

2.1.4 Data compression and chunking for Deep learning

Hailey and Arms [14] investigate strategies for addressing the issues of efficiently storing huge datasets using HDF5, a common data format for high-performance scientific computing. The strategies mentioned in regards of using data compression and chunking, researchers and engineers may overcome storage constraints and efficiently handle CAD data throughout the conversion process in order to overcome storage constraints. .

2.2 Related Tools and Libraries

2.2.1 h5py for file conversion

The CADConverter algorithm utilizes a Python library, h5py which is essential for file and directory handling, and parallel processing. h5py [15] is a Python module that allows you to interact with Hierarchical Data Format version 5 (HDF5) files.h5py is a useful tool for data storage and analysis since it allows users to use Python to generate, read, write, and edit HDF5 files. Users may read and write NumPy arrays[] to and from HDF5 datasets directly, making data sharing between scientific computing tools easier. h5py enables parallel I/O, allowing many processes or threads to read step files and write data at the same time, which can considerably enhance cad conversion algorithm performance.

2.2.2 MeshIO for mesh processing

MeshIO [16] for CAD files is a software application that allows the reading and writing of mesh data in multiple CAD formats. It enables the sharing of geometric information, connection, and other features between CAD programs in real time. MeshIO facilitates interoperability by translating several CAD mesh formats (e.g., OBJ, STL, STEP) into a uniform representation, allowing for fast mesh processing and visualisation. It's a useful tool for engineers and designers since it provides a standardised means of working with CAD meshes, guaranteeing easy data transmission and increasing efficiency in CAD-related jobs.

2.3 Challenges and Opportunities in Cad files conversion

This assessment of the literature offers a solid foundation for the project; nonetheless, there are various obstacles and possibilities in the field of geometry processing that merit additional investigation.

Scalability is a major issue for geometric deep-learning models in the CAD sector. For

efficient training, these models require vast volumes of data, and getting large-scale CAD datasets may be challenging and time-consuming. Furthermore, the large computing resources required for training and inference may make some CAD translation applications difficult to execute.

Despite these obstacles, investigating the use of geometric deep learning [17] in CAD file translation can lead to game-changing advances in the industry. Geometric deep learning has the potential to dramatically increase the efficiency and accuracy of CAD file conversion procedures with continuous research and advances in data accessibility and model scalability.

2.4 Project Overview and Connection to Background Work

The present project is focused on the creation and implementation of several algorithms with the goal of effectively transforming CAD data into an open-source format. By taking this approach, the project hopes to address the obstacles associated with fragmented, interoperable, and inflexible CAD systems, therefore improving CAD information accessibility and flexibility across machine learning and industrial applications. The project's major goal is to optimise CAD file translation and processing efficiency while minimising data loss. The project aims to expedite CAD file conversion by utilising open-source formats and innovative algorithms, making it simpler for different CAD systems to connect and share information easily. Finally, the goal of this study is to contribute to a more coherent and integrated CAD environment so that data can be used for machine learning models.

The Background Work chapter offers the framework for prospective solutions and to improve the conversion of CAD step files to the more efficient and adaptable HDF5 format by addressing the issues encountered in geometry processing. The knowledge gained illuminates the gaps and potential for development in CAD data translation. Overall, the Background Work chapter is critical in steering projects to successful completion while also enhancing the state-of-the-art in CAD data interchange and processing.

Chapter Three

Preliminaries

The next parts provide background information and essential terms that assist in better comprehension of the project. Section 3.1 contains a dictionary of significant terminology and its definitions as they appear in this study. The Section 3.2 provides the technical background for this project.

3.1 Definitions and Terminology

- Geometry [18]: Geometry is a mathematical discipline that studies and analyses the characteristics of geometric forms such as lines, curves, points, and surfaces. It investigates their connections, measures, and changes. CAD geometry data Step files are essential in many engineering and design applications, allowing engineers, architects, and designers to effectively see, analyse, and edit complex 3D objects. Overall, Computational geometry is the study of algorithms and data structures for geometrically formulated problems.

- Topology [19]: Topology is concerned with the qualities of geometric objects that stay intact during deformations such as stretching and bending while retaining their basic features. A topological file specifies the relationships between the various form components. There is information about edges, faces, loops, and shells. It specifies

how geometric objects are connected, such as which points are joined by edges and which edges form faces or surfaces.

- 2D Curves [20]: 2D curves are often used in CAD files to depict planar features such as drawings, profiles, and cross-sections. When converting CAD files to HDF5 format, it is critical to appropriately extract and save these 2D curves in order to keep the geometrical information. Straight lines, circles, ellipses, parabolas, and hyperbolas are some of the examples.

- 3D Curves [19]: Maintaining the structure and features of 3D curves, such as Bezier curves, helixes, and B-spline curves, during CAD file conversion is critical to ensuring that the final HDF5 representation faithfully captures the spatial information of the original design. Bezier curves and B-spline curves are examples of 3D curves.

- Vertices [21]: Vertices are basically three-dimensional points defined by their precise coordinates. They are the basis for modelling complicated geometries

- Faces [22]: Faces are 3D surfaces in CAD models that define the exterior and interior borders of objects. They can be flat or curved, and their shape and contour are determined by their bounding loops.

- Halfedges [21]: Halfedges are directed edges that hold important topological information such as object orientation and adjacency connections. Halfedges are important in CAD conversion because they assist build links between faces, edges, and vertices, assuring the CAD model's right connectivity.

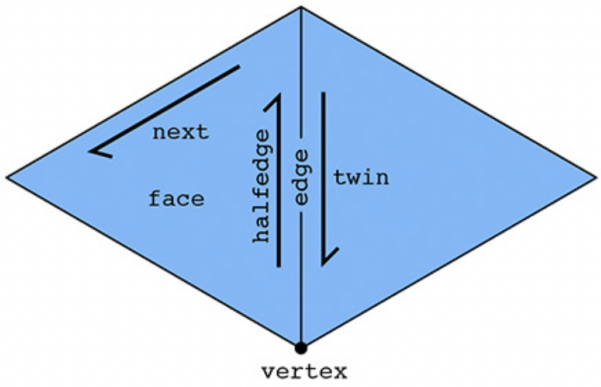

Figure 3.1 Vertex, Halfedge and edge

- STEP files [4]: STEP files are intended to be a universal standard for representing different elements of a product's geometry, topology, and structure. STEP files include geometric data about 3D models such as curves, surfaces, and solid bodies. This data is represented mathematically using entities such as B-splines and NURBS, allowing complicated forms to be precisely represented. Its approval as an ISO standard provides data representation consistency and uniformity, making it a popular alternative for data sharing.

- YAML files [23]: YAML files are used to save configuration parameters including file locations, conversion settings, and STEP file processing choices. Users can customise the conversion procedure without changing the underlying code. YAML files can be used to store metadata about STEP files, such as their names, attributes, or version.

- Fusion360 Dataset [5]: The Fusion360 dataset consists of 2D and 3D models and related data created and maintained using Autodesk's Fusion 360 program. Fusion 360 is a robust cloud-based CAD/CAM platform that is extensively utilised in a variety of sectors for product design, engineering, and production. The collection includes a wide range of 3D models, including components, assemblies, and drawings, that were built with Fusion 360's user-friendly and feature-rich tools[25].

- Assembly Dataset [24]:Assemblies are a useful tool for structuring and organising elaborate ideas, allowing engineers and designers to reduce large systems down into manageable components. Each assembly part relates to a specific design element, such as a component, sub-assembly, or even a fastener. The total assembly correctly captures the spatial arrangement and interactions between its individual pieces by expressing these elements as 3D forms.

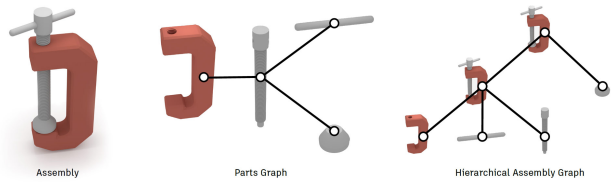

Figure 3.2 Assembly Dataset from [24]

- Reconstruction Dataset [25]:The Reconstruction Dataset comprises sequential design data from a subset of simple 'sketch and extrude' components that allow for the reconstruction of final geometry.

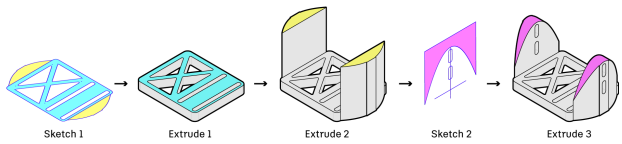

Figure 3.3 Reconstruction Dataset from [25]

- Segmentation Dataset [26]:A segmentation dataset is a collection of data that has been separated or segmented into various segments or components based on particular criteria in Fusion 360. This method entails analysing and breaking down a complicated model or design into discrete parts or regions, each of which represents a different section or component within the larger structure.

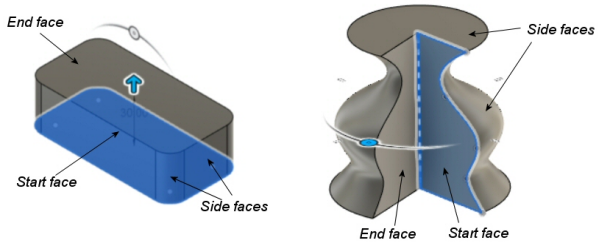

Figure 3.4 Segmentation dataset from [26]

3.2 Technical Background

The information needed to better comprehend this project, including technical details like the HDF5 FILES,Compute Canada and B-splines. Additional reading materials are included.

3.2.1 HDF5 FILES

Understanding the end output of the conversion procedure requires familiarity with the HDF5 format. HDF5 is a data model, library, and file format that provides for the organised and efficient storage of big and complex data sets.

3.2.2 B-splines

In geometric modelling, B-splines are essential for expressing curves and surfaces. B-splines are piecewise polynomial functions with a set of control points and degrees that govern the smoothness of curves or surfaces. These control points enable the depiction of complicated forms to be flexible and precise. Also, Non-Uniform Rational B-Splines (NURBS) [27] are a kind of B-spline that is commonly utilised in CAD programmes. NURBS have weights for each control point, which improves control over the form of curves or surfaces. This capacity is very useful in CAD conversion operations, where exact representation and smoothness are required to transform complicated 3D objects properly.

3.2.3 Compute Canada

Compute Canada [28] is the country's high-performance computing (HPC) system. The system provides users with access to both storage and computation capabilities, allowing them to execute huge data analyses more efficiently than they could on a typical desktop computer. We have used Compute Canada account configured using the university set-up and performed the batch processing of cad step files and validated the CADConversion algorithm. Researchers in the subject of CAD file conversion may be able to use Compute Canada's computational capabilities to speed simulations, optimise methods, or handle massive datasets relevant to CAD conversions.

Chapter Four

CAD Data

4.1 Introduction

This chapter seeks to offer a complete description of the data utilised in the CadConversion Process. This chapter will concentrate on the processing of step data and process to convert step files to YAML as well as the conversion to the proposed HDF5 file format, while the preceding chapter will offer the essential foundation knowledge.

4.2 Step Data and Yaml Data

For computer aided design(CAD),Step files has been universally accepted.CADConverter algorithm first process the step files and extracted the data needed for file conversion to HDF5 format.

The Step Processor class uses Open CASCADE Technology (OCC) libraries and custom modules to handle STEP files. It reads the STEP file, extracts the sections, and, if necessary, transforms them to NURBS [27] surfaces. It heals shapes before extracting topology, geometry, statistics, and meshes for each portion. Data is extracted and stored in dictionaries before being written to YAML files in the output directory. The class enables customised processing choices and the ability to handle various data types. The provided code defines a

Python class called StepProcessor, which processes STEP files (Standard for the Exchange of Product Data). The StepProcessor is designed to extract various information from STEP files, including geometry, topology, statistics, and mesh data. It relies on various libraries and custom modules to handle the processing tasks.The key components and functionalities of the StepProcessor class.The init method initializes the stepProcessor object. It takes several parameters, including the path to the STEP file (stepfile), the output directory for processed data (outputdir), the log directory for storing log files (logdir), and optional custom classes for entity mapping (entitymapper), topology building (topologybuilder), geometry building (geometrybuilder), mesh building (meshbuilder), and statistics building (statsbuilder).

Geometric information about the 3D shapes, such as vertex coordinates, surface information, and curve data. Mesh data (meshes): Surface meshes are created for the CAD parts using the mesh builder. Statistical data (statsdict): Additional statistical information related to the CAD entities and their characteristics.

4.3 Proposed HDF5 CAD Files' Structure

- Geometry Data:The geometry file defines the main parameters for both the 2D and 3D versions, including the centre, radius, and axis vectors. It also includes bounding box information, surface information, and a list of vertices.

- Topology Data :A topological file defines the connections between the various components of a form. Edge information, face definitions, loops, and shells are all included. It defines how geometric things are linked together, such as which points are joined by edges and which edges create faces or surfaces.

- Mesh Data:The word "mesh" in an HDF5 (Hierarchical Data Format version 5) CAD file refers to the geometric representation of the 3D surfaces that comprise the CAD model. HDF5 is a flexible file format that may hold a variety of data types such as

numerical data, pictures, metadata, and more. When applied to CAD files, HDF5 serves as an appropriate container for storing 3D model mesh data. The mesh data contained within an HDF5 CAD file contains critical information on the vertices, edges, and faces that determine the geometry of the 3D object. Each vertex corresponds to a single point in 3D space, whereas edges link vertices, thus establishing the surfaces' bounds. As a result, the faces form as combinations of these edges, finally sculpting and defining the CAD model's surfaces.

CAD software can quickly access, manage, and visualise complex 3D models by utilising HDF5 for mesh storage. Furthermore, HDF5's hierarchical structure allows for the insertion of extra data, such as metadata and characteristics, which enhances the representation and sharing possibilities of CAD models. Here Group IDs are used to organise and group items in mesh CAD data, Points represent individual vertices in 3D space, and Triangles are the fundamental building blocks used to form surfaces in the mesh model. These components, when combined, constitute the basis for representing 3D objects in computer graphics and engineering applications. Also, Group IDs may be used to organise and manage distinct portions or areas of the mesh, allowing for more effective data handling and analysis.

- meshPath option is a directory path that corresponds to the location where Wavefront OBJ mesh files are kept. An OBJ file is a common format for encoding 3D models. It provides information on the 3D vertices (points), edges, and faces that determine the geometry of the 3D model.It uses the'meshio' library to read the Wavefront OBJ mesh files from the specified'meshPath'. The'meshio' library is a Python library that can read and write mesh file types such as OBJ. 'meshio.read()' reads each OBJ mesh file one by one, retrieving the following data for each file:

  1. points: The vertex coordinates that define the mesh's 3D points.
  2. cells: Data on the connection of vertices to construct cells or elements such as

triangles, tetrahedra, and so on.

3. celldata: Extra information for each cell, such as material attributes or group IDs.

- Storing mesh data in subgroups inside the mesh group: The function saves the mesh data in the HDF5 file's'mesh' group. It establishes different subgroups inside the'mesh' group to organise data for numerous meshes. To differentiate distinct meshes, each subgroup is given a unique name with a zero-padded index (e.g., '000', '001', '002').

- Mesh data storage as HDF5 datasets: The function saves the extracted mesh data as HDF5 datasets within each mesh subgroup. Specifically,The vertex coordinates are recorded in a collection called 'points'. Cell data offers information on the connectivity of vertices for each cell type (for example, triangles and tetrahedra). The programme generates distinct datasets for each cell type, with names that correspond to the cell type (e.g., 'triangle', 'tetra'). Cell_data provides extra information about each cell, such as material attributes and group IDs. The code determines whether the data is a single value or a set of values. If there is just one value, it is saved as a dataset within the subgroup.If it's a dictionary, the code creates a subgroup called "key" and keeps the dictionary values as independent datasets within that subgroup.

  Following this technique, the function effectively organises and saves the mesh data in the HDF5 file, allowing for simple access and retrieval of the geometric and extra information of the 3D model. However, keep in mind that the supplied code snippet makes use of various functions (convert_dict_to_hdf5, convert_stat_to_hdf5, meshio. read) that are not specified in the sample, and their implementation would have a substantial influence on the overall functioning and completeness of the code.

- Stat Data- The "stat" component in Computer-Aided Design (CAD) refers to the statistical data connected with the CAD model. Beyond its geometric and topological representation, it contains extra information and features that give useful insights and

characteristics of the 3D object.Stat file has been structured as following-

1. Singularities:In the context of geometrical forms, singularities are points, edges, or areas where the shape shows distinctive or unique qualities that generally deviate from normal behaviour. Singularities can emerge as a result of form limitations or symmetries, resulting in non-standard features.
2. Exact_domain :In geometry, the exact_domain is the parameter range that defines a certain geometric form or curve. It defines the valid values for the parameters that define the shape.
3. Has_singularities :The boolean attribute "has_singularities" shows whether a geometric form or curve has singularities. If this trait is true, it indicates that the form has points, edges, or regions that depart from typical behaviour, suggesting unique or exceptional properties.
4. Nr_singularities:The term "Nr_singularities" refers to the number of singularities in a geometric form or curve. It denotes the number of points, edges, or areas where the form shows unique traits that differ from its overall behaviour.
5. Outer_wire:An "outer_wire" is a closed loop or border that forms the outermost boundary of a geometric shape or surface in the context of geometrical modelling. It is important in establishing the general form and topology of the thing. The outside wire encloses the shape's interior and acts as the fundamental boundary for all subsequent geometric operations or analysis.

Chapter Five

System Implementation

This chapter describes in detail how the CADConverter Algorithm works. The chapter is split into four sections. First, in Section 5.1, we examine the programming language, libraries, and frameworks that we utilised, as well as why we chose them. Section 5.2 then delves into the implementation process using code samples and pseudocode. In the following chapter, we will go through the testing technique we used to ensure that the algorithm works correctly and effectively. We also discuss how we improved the algorithm's performance and present our findings.

5.1 Programming Language and Tools

For various convincing reasons, we chose the Python[29]programming language for our project implementation. To begin, Python is notable for its straightforward and intuitive syntax, which makes code very readable and manageable. Second, Python has a wide choice of libraries that cater to diverse scientific and technical areas, greatly simplifying the development process. Python has also gained extensive recognition in scientific and technical sectors, demonstrating its dependability and efficacy. The language's growing ecosystem provides us with an assortment of scientific computing tools, allowing us to accelerate development and ensure rapid code execution. In the implementation phase, the following

libraries and frameworks were used:

1. NumPy: is a robust Python package that provides a high-performance array object that is extremely useful for numerical operations. It provides a variety of tools for effectively working with these arrays, making complicated mathematical processes simpler. NumPy is a go-to tool in the realm of scientific computing, whether you're working with enormous datasets or conducting quick array manipulations. [30]

2. SciPy: SciPy is the Python library to use if you need to execute complex scientific computations and handle mathematical functions. This library contains modules dedicated to optimisation, linear algebra, integration, interpolation, and many more functions. It's a flexible and useful tool for researchers, engineers, and anybody else who wants to tackle complicated mathematical issues in Python applications.

3. HDF5 - The HDF5 software library and its related tools are open-source and publicly available, allowing them to be used and developed by a large community of users and developers. The programme would then extract and transform the required data, such as 3D model geometry, surface information, and other properties, into an HDF5-compatible structure. The HDF5 format supports sophisticated hierarchical data representation, making it excellent for storing CAD models, their characteristics, and associated information. [10]

4. Mesh.IO-Meshio allows users to import mesh data from formats such as STL, OBJ, OFF, Gmsh, and others, as well as export mesh data to other formats. This adaptability allows for easy data transmission between various applications and analysis tools that may employ different mesh formats. The library is especially useful in computer-aided design (CAD) applications, where mesh data is used to define geometries and simulations. [16]

5. Sympy: is a Python library for symbolic mathematics. It enables the manipulation of

mathematical expressions, the execution of calculus and algebraic operations, and the symbolic solution of equations. [31]

6. Meshplot:Meshplot is a Python library that allows you to visualise 3D mesh data. It has an easy-to-use interface for plotting and rendering 3D objects. [32]

7. PyYAML : PyYAML is a Python package that makes it easier to interact with YAML files in Python programmes. YAML is a data serialisation format that is human-readable.

8. Munch: A Python library that provides a dictionary-like object called a "Munch" with access to its keys through attributes. It makes working with nested data structures more convenient.[33]

5.2 Execution

Because the code is organised into modules and classes, it is easy to update and maintain.The primary modules are as follows:

1. Pre Procesisng- :The 'process_files' function is used to process the transformed files further. It accepts as input the list of successful files ('success'), the output directory path, the HDF5 file path, and the batch and job IDs. This function organises and translates CAD data to the HDF5 format, as well as handles post-processing chores.

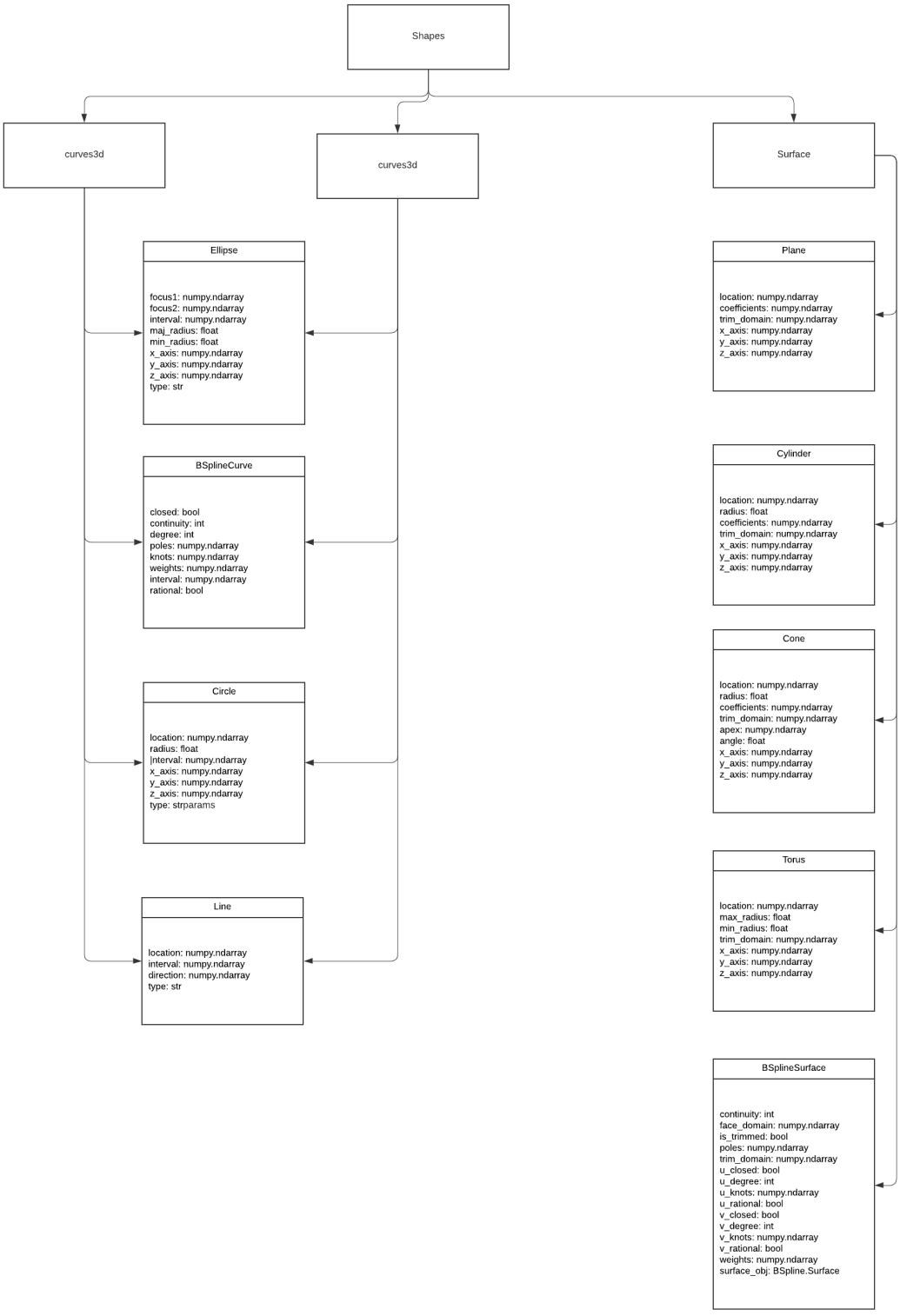

Figure 5.1 Class diagram of Shapes used in CadConverter Project

This module plays a crucial role in file conversion. It efficiently turns the data to the HDF5 format using the 'convert_data_to_hdf5' function, assuring seamless and organised conversion. It reads the contents of these YAML files and stores them as Python dictionaries in the variables 'geometry_data,' 'topology_data,' and'stat_data,' correspondingly, using the 'load_dict_from_yaml' function. These dictionaries will include the necessary information for the script's later processing and conversion of CAD files.

2. Code Refactoring-: Several refactoring approaches were used to improve the overall quality of the code. Math operations in the shape classes were simplified by utilising NumPy arrays and functions, which reduced unnecessary computations and made the code more succinct. The following are some of the code refactoring changes:

   (a) Extracting Functions: The code was broken down into smaller functions, each with its own responsibilities. To handle particular file loading and conversion duties, functions such as load_dict_from_yaml,load_dict_from_hdf5, and convert_hdf5_to_dict were removed.

   (b) Using Meaningful Variable and Function Names: To make the code more self-explanatory, meaningful variable and function names were utilised. For example, classes such as Line, Circle, Ellipse, and so on were designed to represent various shapes, boosting code readability.

   (c) Eliminating Duplicate Code: Duplicate code was eliminated by combining similar functionality into reusable routines. For example, equivalent sample and derivative methods were defined in classes such as Line, Circle, and Ellipse, minimising repetition. Instead of many if-else conditions for data formats in the load_dictionary_from_file function, the function now calls the necessary format-specific methods directly. This increases the clarity and maintainability of the code.

3. File Conversion-: The given script runs through STEP files in a directory. It parses command-line arguments using the argparse library, enabling users to provide input, output, log, and HDF5 file locations, as well as job and batch IDs.

   The script begins by creating an argument parser with descriptions for each parameter. The command-line parameters are then processed with 'parser.parse_args()', and the results are placed in the 'args' variable.The script then runs the function 'process_step_files' from the 'cadmesh.utils.processing' module, giving the input and output folders as well as the log path. This function processes the STEP files in the input directory and produces two lists:'success' with the names of successfully processed files and 'failed' with the names of files that failed to process.

4. Conversion to HDF5 Format–: The provided Python script performs data conversion for a complex 3D shape from YAML and OBJ files to an HDF5 file. The script uses h5py and media libraries for handling HDF5 and mesh data, and the yaml library for parsing YAML files. The conversion process is organized into functions that recursively traverse the data dictionaries and convert them into appropriate HDF5 data structures.

5. Accelerated CAD Batch Processing for Efficient Results -: To perform the .step files conversion to YAML and HDF5, this Bash script uses the SLURM task scheduler to handle STEP CAD files in a high-performance computing environment. It works as a batch job with several jobs, handling 100 tasks at the same time. When the script is finished, it outputs "Done!" This solution enables effective parallel translation of CAD data into YAML and HDF5 formats on an HPC system.

All the modules are further divided into different classes. We will go through the definitions of each class below starting from processing parts of step files to the batch processing of converted files using Compute Canada.

5.2.1 Step Data extraction

In general, the process_parts function is critical in the CAD data processing workflow. It guarantees that each component is accurately translated, repaired (if necessary), and processed in order to retrieve critical information Its primary purpose is to handle the processing of individual parts within a CAD file in the STEP format. Here we have a Python script that processes sections of STEP-formatted CAD files.

Here's the process_parts file method implementation:

```
def process_parts(self, convert=False, fix=False, write_face_obj=True,
    write_part_obj=True, indices=[], version="2.01"):
        if len(self.parts) == 0:
            self.logger.info("No parts loaded to process.")
            return

        # If no indices are given, process all parts
        if len(indices) == 0:
            indices = range(len(self.parts))
            self.logger.info("Processing all %i parts of file."%len(indices))

        topo_dicts = []
        geo_dicts = []
        mesh_dicts = []
        stats_dicts = []

        # Iterate over all indices
        for index in indices:
            part = self.parts[index]
```

```
# Convert complete part to NURBS surfaces
if convert:
    try:
        nurbs_converter = BRepBuilderAPI_NurbsConvert(part)
        nurbs_converter.Perform(part)
        part = nurbs_converter.Shape()
    except Exception as e:
        #print("Conversion failed, processing unconverted")
        #print(e.args.split("\n"))
        self.logger.error("Nurbs conversion error:
            %s"%"".join(str(e).split("\n")[:2]))
        continue

# Fix shape with healing operations
if fix:
    #print("Fixing shape")
    b = _ShapeFix_Shape(part)
    b.SetPrecision(1e-8)
    #b.SetMaxTolerance(1e-8)
    #b.SetMinTolerance(1e-8)
    b.Perform()
    part = b.Shape()

# Extract information for part
try:
    topo_dict, geo_dict, meshes, stats_dict = self.__process_part(part)
except:
    self.logger.error("Processing part failed %i"%index)
```

```
            continue

        topo_dicts.append(topo_dict)
        geo_dicts.append(geo_dict)
        stats_dicts.append(stats_dict)

        if write_face_obj:
            mesh_path = self.output_dir / f"{self.step_file.stem}_mesh"
            #print(str(mesh_path), mesh_path)
            os.makedirs(mesh_path, exist_ok=True)
            for idx, mesh in enumerate(meshes):
                if len(mesh["vertices"]) > 0:
                    igl.write_triangle_mesh("%s/%03i_%05i_mesh.obj"%(str(mesh_path),
                        index, idx), mesh["vertices"], mesh["faces"])

    # Write out dictionary lists
    self.logger.info("Writing dictionaries")
    topo_yaml = self.output_dir / f"{self.step_file.stem}_topo"
    write_dictionary_to_file(topo_yaml, {"parts": topo_dicts, "version":
        version}, self.data_format)
    self.logger.info("Topo dict: Done")
    geo_yaml = self.output_dir / f"{self.step_file.stem}_geo"
    write_dictionary_to_file(geo_yaml, {"parts": geo_dicts, "version":
        version}, self.data_format)
    self.logger.info("Geo dict: Done")
    stats_yaml = self.output_dir / f"{self.step_file.stem}_stat"
    write_dictionary_to_file(stats_yaml, {"parts": stats_dicts, "version":
        version}, self.data_format)
```

```
        self.logger.info("Stat dict: Done")
```

The function begins by determining whether there are any pieces to process. If no parts are found, it records a message and exits.If no indices are specified, the function will process all of the components in the CAD file.Depending on the inputs (convert and fix), the function can possibly convert each part to NURBS surfaces and conduct shape fixing.Following any required conversions and repairs, the function uses the process_part method to retrieve information from the part. This data comprises topology, geometry, statistical information, and 3D mesh data.The 3D mesh data for faces and parts is written to separate OBJ files if provided (write_face_obj and write_part_obj).This function is in charge of processing the CAD file's parts. It takes different options such as 'convert,' 'fix,' 'write_face_obj,' 'write_part_obj,' 'indices,' and 'version' to control the processing. Each part's extracted information is saved in its own dictionary (topo_dicts, geo_dicts, mesh_dicts, and stats_dicts). This script uses various builders and mappers to extract information from the CAD parts, including topology, geometry, statistics, and meshes.Finally, the dictionaries that hold the information for all of the sections are written to YAML files with the proper filenames (_topo, _geo, and _stat). The filenames are generated from the original STEP file name.

5.2.2 Step File Processing

The 'process_step_files Python function prepares CAD files for conversion to HDF5 format. It accepts'success_files,' a list of successful files, as well as'models_folder,' 'output_folder,' 'batch_id,' and 'job_id' as input parameters.The output of this process is written to YAML files with the proper filenames. This script automates the translation of STEP CAD data to HDF5, allowing for faster data processing and analysis in a variety of scientific and engineering applications. It speeds up the process by processing numerous files at once.

1. process_single_step()

   The method accepts a single STEP file as input and converts it to HDF5 format initializes a StepProcessor object (sp) with the given STEP file, output directory, and log directory.

```
def process_single_step(sf, output_dir, log_dir, produce_meshes=True):
    try:
        if produce_meshes:
            sp = StepProcessor(sf, Path(output_dir), Path(log_dir))
        else:
            sp = StepProcessor(sf, Path(output_dir), Path(log_dir),
                mesh_builder=None)
        sp.load_step_file()
        sp.process_parts()
        return sf, None
    except Exception as e:
        return sf, str(e)
```

   Hence, this function loads the STEP file, processes its components, and reports the success status as well as any error messages detected.

2. process_step_files()-This programme runs over a text file containing a list of STEP files (input_file_list).It reads the list of input files from the text file and starts the processing for each file with the process_singlestep function.It creates output files in the outputdir supplied and log files in the log_dir specified.The function produces two lists: success_files, which contains the successfully processed STEP files, and failed_files, which contains tuples of the failure files and their associated error messages.

```
def process_step_files(input_file_list, output_dir, log_dir):
    output_dir = Path(output_dir)
```

```
log_dir = Path(log_dir)

os.makedirs(output_dir, exist_ok=True)
os.makedirs(log_dir, exist_ok=True)

# Read input files from a list in a text file
with open(input_file_list, 'r') as f:
    input_files = [Path(line.strip()) for line in f]

success_files = []
failed_files = []

with tqdm_joblib(tqdm(desc="Processing step files",
    total=len(input_files))) as progress_bar:
    results = Parallel(n_jobs=4)(delayed(process_single_step)(sf,
        output_dir, log_dir) for sf in input_files)

model_names = [] # this will hold just the model names

for sf, error_message in results:
    if error_message is None:
        success_files.append(sf)
        model_names.append(sf[0].name) # append only the name of the file
    else:
        failed_files.append((sf, error_message))

return success_files, failed_files
```

5.2.3 HDF5 Conversion

The function runs through the'success_files', collecting pertinent information such as the model's name and establishing paths for the geometry, topology, statistics YAML files, and the mesh folder. It then changes the data to HDF5 format using the 'convert_data_to_hdf5' function.When the conversion is complete, the function deletes the original files and directories. If the conversion fails, the error information are saved in the 'failed_conversions' list. The function outputs the names of successfully converted and unsuccessful models to separate text files after processing all files. If any conversions fail, an error notice is displayed and the programme terminates with a status code of 1. Otherwise, it says that it has been completed successfully and deletes the model's folder before departing with a status code of 0.

```
def process_files(success_files, models_folder, output_folder, batch_id, job_id):
    successful_conversions = []
    failed_conversions = []
    input_folder = Path(models_folder)
    output_folder_path = Path(output_folder)

    for item in success_files:
        try:
            output_folder_path.mkdir(parents=True, exist_ok=True)
            model_name = item[0].stem

            meshPath = input_folder / f"{model_name}_mesh"
            geometry_yaml_file_path = input_folder / f"{model_name}_geo.yaml"
            topology_yaml_file_path = input_folder / f"{model_name}_topo.yaml"
            stat_yaml_file_path = input_folder / f"{model_name}_stat.yaml"
            output_file = output_folder_path / f'{model_name}.hdf5'
```

```
            assert meshPath.exists()
            assert geometry_yaml_file_path.exists()
            assert topology_yaml_file_path.exists()
            assert stat_yaml_file_path.exists()

            geometry_data = load_dict_from_yaml(geometry_yaml_file_path)
            topology_data = load_dict_from_yaml(topology_yaml_file_path)
            stat_data = load_dict_from_yaml(stat_yaml_file_path)

            convert_data_to_hdf5(geometry_data, topology_data, stat_data, meshPath,
                output_file)

            successful_conversions.append(model_name)
```

The script conducts a status check and cleaning after completing the CAD file processing. If any conversions fail during processing, the script produces a message noting that further information may be found in the 'failed conversions_job_idbatch_id.txt' file and quits with a status code of 1 to indicate an unsuccessful completion. If all files are successfully converted, the script shows the message "All files successfully converted." It then deletes the models folder in the 'input_folder,' resulting in a clean and organised workspace.

```
if len(failed_conversions) > 0:
        print(f"Some files failed conversion. Check
            'failed_conversions_{job_id}_{batch_id}.txt' for details.")
        exit(1)
    else:
        print(f"All files successfully converted. Check
            'successful_conversions_{job_id}_{batch_id}.txt' for details.")
```

```
        try:
            if input_folder.exists():
                shutil.rmtree(str(input_folder)) # Delete the models folder
        except OSError as e:
            print(f"Error: {input_folder} : {e.strerror}")
        exit(0)
```

Finally, regardless of whether any conversions failed or not, the script quits with a status code of 0, indicating that the processing operation was successfully completed.

The script starts by defining the input and output paths for the YAML and OBJ files. It then loads the YAML data into dictionaries using the 'load_dict_from_yaml' function. After verifying the existence of the necessary files and directories, the 'convert_data_to_hdf5' function is called to create the HDF5 file.

'convert_data_to_hdf5'() The 'convert_data_to_hdf5' function takes the loaded geometry, topology, and statistical data, along with the mesh directory, and processes them to generate the HDF5 file. It creates groups for geometry, topology, stat, and mesh in the HDF5 file and recursively converts the data dictionaries into corresponding datasets within the groups.

```
def convert_data_to_hdf5(geometry_data, topology_data, stat_data, meshPath,
    output_file):
    with h5py.File(output_file, 'w') as hdf:
        geometry_group = hdf.create_group('geometry')
        convert_dict_to_hdf5(geometry_data, geometry_group)


        topology_group = hdf.create_group('topology')
        convert_dict_to_hdf5(topology_data, topology_group)
```

```
stat_group = hdf.create_group('stat')
convert_stat_to_hdf5(stat_data, stat_group)

mesh_group = hdf.create_group('mesh')
for index, mesh_file in enumerate(sorted(os.listdir(meshPath))):
    if mesh_file.endswith(".obj"):
        mesh = meshio.read(os.path.join(meshPath, mesh_file))
        points = mesh.points
        cells = mesh.cells
        cell_data = mesh.cell_data

        mesh_subgroup = mesh_group.create_group(str(index).zfill(3))
        mesh_subgroup.create_dataset('points', data=points)
        for cell in cells:
            cell_type = cell.type
            cell_indices = cell.data
            mesh_subgroup.create_dataset(cell_type, data=cell_indices)

        for data_key, data_value in cell_data.items():
            if data_key == "obj:group_ids":
                data_key = "group_ids"
            if isinstance(data_value, list):
                mesh_subgroup.create_dataset(data_key, data=data_value)
            else:
                subgroup = mesh_subgroup.create_group(data_key)
                for field_key, field_value in data_value.items():
                    subgroup.create_dataset(field_key, data=field_value)
```

```
output_file = 'Complex shape v3.hdf5'
```

As a result, It generates an HDF5 file called 'Complex shape v3.hdf5' and groups the CAD data into 'geometry,' 'topology,"stat,' and mesh categories. The function uses auxiliary functions for each CAD component, allowing for easy data translation and storage. The generated HDF5 file may now be used for further analysis and visualisation with relevant tools and libraries.Overall, the script provides a comprehensive approach to convert complex 3D shape data from YAML and OBJ format to HDF5, making it suitable for efficient storage and manipulation of CAD data.

main function()-Upon execution, the script runs the cadmesh.utils.processing module's process_step_files function, which reads the list of STEP files from the supplied text file, processes them, and stores the results in the output and log folders.The script then invokes the process_files function, which accepts the list of successfully processed files, as well as the output and HDF5 file paths, Batch ID, and Job ID, and converts the data to the HDF5 format. Conversions that are successful are saved in the selected HDF5 file, while failures are noted in the log directory.

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Process STEP files in a
        directory.")
    parser.add_argument("--input", help="Path to the text file with the list of
        STEP files.")
    parser.add_argument("--output", help="Path to the directory where results will
        be saved.")
    parser.add_argument("--log", help="Path to the directory where logs will be
        saved.")
    parser.add_argument("--hdf5_file", help="Path to the HDF5 file where results
```

```
        will be saved.")
    parser.add_argument("--jobId", help="Job ID for this execution")
    parser.add_argument("--batchId", help="Batch ID for this execution")
    args = parser.parse_args()

    success, failed = cadmesh.utils.processing.process_step_files(args.input,
        args.output, args.log)

    process_files(success, args.output, args.hdf5_file, args.batchId, args.jobId)
```

In summary, this script serves as the primary entry point for converting STEP files to the HDF5 format, allowing for input/output route flexibility, log storage, and batch processing for CAD data conversion.

5.2.4 Streamlined CAD Batch Processing in STEP Format

The script's main point is to run a Python conversion script (cloud_conversion.py) that is in charge of processing the CAD files in the batch directory. In the output and HDF5 paths, the conversion creates YAML and HDF5 files, respectively. The script configures SLURM task parameters such as input and output routes, batch ID, and job ID, activates a Python environment, and specifies input, output, log, and HDF5 file directories. Here's the load_yaml_file method implementation:

```
#!/bin/bash
#SBATCH --time=2:00:00
#SBATCH --account=def-teseo
#SBATCH --job-name=step-process
#SBATCH --mem-per-cpu=32G
#SBATCH --cpus-per-task=2
```

```
#SBATCH --array=0-99



# Activate Python environment
echo "Activating Python environment..."
module load python/3.8.10
source ~/scratch/sachin/cad/bin/activate

# Define paths relative to the scratch directory
echo "Defining paths..."
DATA_PATH="/home/sachin/scratch/sachin/Fusion360/segmentation/step/s2.0.1_extended_step/breps/st
OUTPUT_PATH="/home/sachin/scratch/sachin/Fusion360/segmentation/yaml/s2.0.1_extended"
LOG_PATH="/home/sachin/scratch/sachin/Fusion360/segmentation/yaml/logs"
HDF5_PATH="/home/sachin/scratch/sachin/Fusion360/segmentation/hdf5"

BATCH_ID=$SLURM_ARRAY_TASK_ID # BatchID is equal to the task id in the job array
JOB_ID=$SLURM_JOB_ID # JobID from SLURM

# Define new path for this batch
echo "Defining new path for this batch..."
BATCH_PATH="$DATA_PATH/batch_$BATCH_ID"
mkdir -p "$BATCH_PATH"

# Get the list of files to be copied
echo "Getting the list of files to be copied..."
FILES_TO_COPY=$(ls -1 "$DATA_PATH"/*.stp | sed -n
    "$((SLURM_ARRAY_TASK_ID*500+1)),$((SLURM_ARRAY_TASK_ID*500+500))p")
```

```
# Check if files exist
if [ -z "$FILES_TO_COPY" ]
then
echo "No files to copy for batch $BATCH_ID."
exit 1
fi

# Copy the files
echo "Moving the files..."
mv $FILES_TO_COPY "$BATCH_PATH"



# Conversion scripts
echo "Running conversion scripts..."
python cloud_conversion.py --input "$BATCH_PATH" --output "$OUTPUT_PATH" --log
    "$LOG_PATH" --batchId "$BATCH_ID" --jobId "$JOB_ID" --hdf5_file "$HDF5_PATH"

if [ $? -eq 0 ]
then
echo "Deleting batch files..."
rm -rf "$BATCH_PATH"
else
echo "Some files were not converted successfully. Not deleting batch files."
fi

echo "Done!"
```

Each task is identified by a BATCH_ID, and the corresponding JOB_ID is also defined

based on SLURM parameters. For each task, a new directory (BATCH_PATH) is created within the data input path to accommodate the files specific to that task. The script then retrieves the list of files (FILES_TO_COPY) meant for processing, depending on the task ID. If no files are present for a particular task, the script gracefully exits.

It generates a separate directory for each job to contain the necessary files. For each task, a new directory (BATCH_PATH) is created within the data input path to accommodate the files specific to that task. The script copies files to the batch directory before running a Python conversion script to create YAML and HDF5 files. Successful conversions result in batch file deletion, whereas failures keep the batch files for debugging purposes.

Chapter Six

Evaluation and Results

This chapter provides an overview of the testing methodologies used for the CADConverter algorithm. This chapter is divided into four sections. Section 6.3 mentions the processes used for the validation of algorithms accuracy and efficacy. Section **??** describes the techniques for optimizing the performance of the algorithm. Section 6.4 discusses the challenges during process implementation, and Section 6.5 shows snapshots of models developed using CADConverter algorithm.

6.1 Conversion's Results

Batch processing results on Canada compute-

2023-07-09 11:25:25,774 INFO Init step processing:120964_7878728b_0000_0003

2023-07-09 11:25:25,939 INFO Loaded parts: 1

2023-07-09 11:25:25,939 INFO Processing all 1 part of file.

2023-07-09 11:25:25,939 INFO Entity mapper: Init

2023-07-09 11:25:25,948 INFO Entity mapper: Done

2023-07-09 11:25:25,948 INFO Extract topo: Init

2023-07-09 11:25:25,948 INFO Extract topo: Build

2023-07-09 11:25:25,955 INFO Extract topo: Done

2023-07-09 11:25:25,955 INFO Extract geo: Init

2023-07-09 11:25:25,955 INFO Extract geo: Build

2023-07-09 11:25:25,964 INFO Extract geo: Done

2023-07-09 11:25:25,964 INFO Extract stats: Init

2023-07-09 11:25:25,965 INFO Extract stats: Done

2023-07-09 11:25:25,965 INFO Extract mesh: Init

2023-07-09 11:25:26,004 INFO Meshing body: Done

2023-07-09 11:25:26,036 INFO Extract mesh: Done

2023-07-09 11:25:26,048 INFO Writing dictionaries

2023-07-09 11:25:26,075 INFO Topo dict: Done

2023-07-09 11:25:26,169 INFO Geo dict: Done

2023-07-09 11:25:26,189 INFO Stat dict: Done

6.2 Sampling Results

The CADConverter Algorithm can handle complicated geometries such as a shocker model in addition to simple forms. Fig 6.1, Fig 6.2 and Fig 6.3 illustrate several perspectives of the shocker model, illustrating how the algorithm can properly sample complicated geometries with HDF5 CAD data.Once the conversion is done, this algorithm returned a set of HDF5 files.Using the SMARTCAD project [34] by our team, we tried to sample the converted file and the following shapes has been sampled for shocker model.

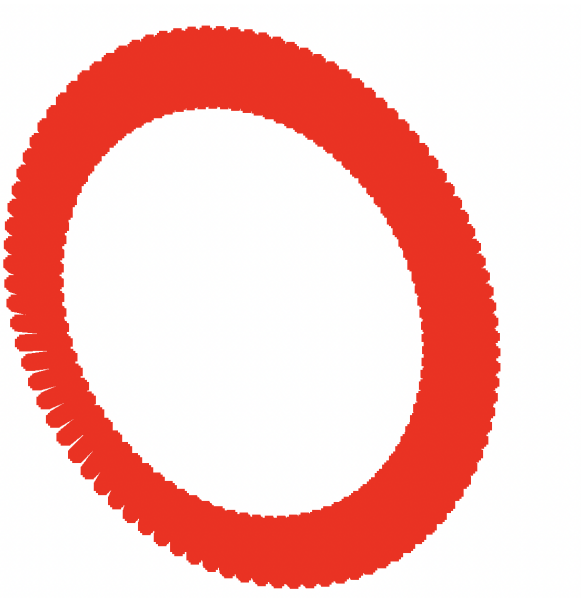

Figure 6.1 Shape sampling of one layer of a shocker

Figure 6.2 Shape sampling of Arc and side view of a shocker

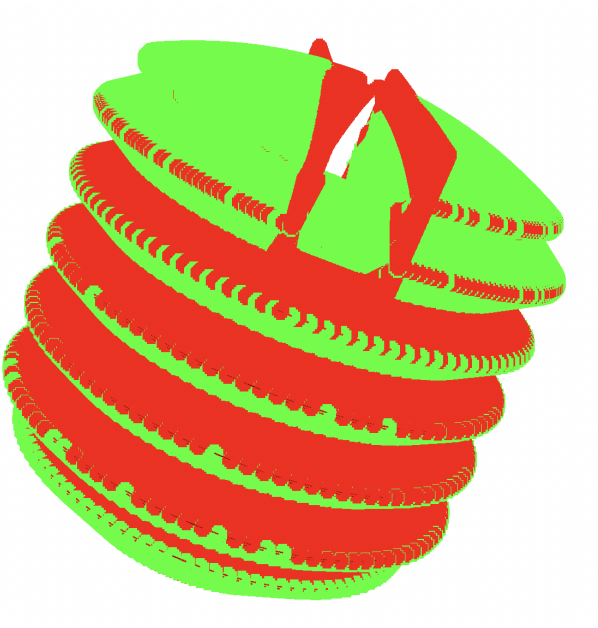

Figure 6.3 Shape sampling of top view of a shocker

6.3 Testing

Testing is essential in the creation of any algorithm in order to confirm its correctness and efficiency. Several tests are carried out to assure effective file conversion-.

1. Boundary Testing: Boundary testing checks the behaviour of the conversion process at the boundaries of valid input data. It examines how the system handles extreme or boundary values and guarantees that the conversion is resilient in these circumstances.

2. Data Integrity Testing:

   This ensures that the converted data in HDF5 format appropriately represents the original CAD model. It ensures that no data is lost, corrupted, or altered during the conversion.

3. Testing at the Unit Level:

Individual conversion functions and methods in charge of handling certain components of the conversion process should be tested.We checked that each function generates the required result for a variety of test cases and settings.To guarantee robustness, we examine edge cases and boundary conditions.For shapes like torus,bsplines,cylinder,cone,sphere etc, we performed unit testing using class TestBasic.

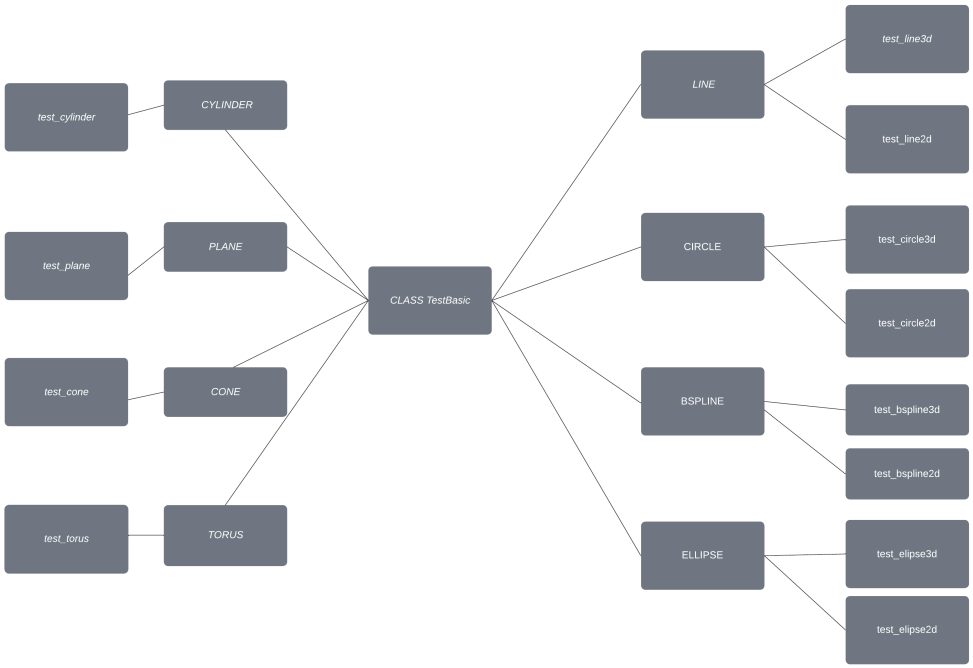

Figure 6.4 Unit Testing by classes of different shapes

For shape line 3d,function test_line3d has been implemented as-

```
class TestBasic(unittest.TestCase):


    def test_line3d(self):
        data = {
            "type": "Line",
            "location": [0, 0, 0],
```

```
    "direction": [1, 0, 0],
    "interval": [0, 1]
}

line = Line(Munch.fromDict(data))
self.assertEqual(line._location.shape, (1, 3))
self.assertEqual(line._direction.shape, (1, 3))
self.assertEqual(line._interval.shape, (1, 2))
sample_points: None = np.linspace(0, 1, 10).reshape(-1, 1)

# sampling
self.assertEqual(line.sample(sample_points).shape, (10, 3))

# derivative shape
self.assertEqual(line.derivative(sample_points, 0).shape, (10, 3))
self.assertEqual(line.derivative(sample_points, 1).shape, (10, 3))
self.assertEqual(line.derivative(sample_points, 2).shape, (10, 3))

d, d2 = curves_derivative(line, sample_points)
self.assertTrue(d < 1e-7)
self.assertTrue(d2 < 1e-7)
```

In the meantime, our team has already been working on another project [34] in which we sampled different datasets containing a variety of shapes such as boxes, circles, cones, cylinders, and ellipses. We successfully completed the necessary tests, including the Surface Patch and Loop Adjacency Tests, as well as the Hausdorff distance evaluation. These tests proved the 3D models' correctness and data integrity for the sample shapes.

6.4 Challenges and Solutions

During the implementation process, we faced several challenges, some of which are listed below:

1. Managing a wide range of complicated geometry and intricate assemblies-Converting CAD STEP files to HDF5 format can be time-consuming and resource-intensive, especially for big assemblies or complicated models.

2. Huge data set-We have tested the CAD conversion algorithm on a huge dataset that comprises ABC dataset as well as Fusion dataset.Total step files in Fusion 360 dataset is around 256,6168.Handling such large number of files was resulting in increasing memory constraints and more processing time.

3. Handling runtime and timeout errors during the conversion.

4. Compute Canada unexpectedly deleted the scratch directory holding our converted data and Python environment during scheduled maintenance.

6.5 Code Refinement-

In this section, we present visualizations generated using the SmartCAD Algorithm to demonstrate the algorithm's effectiveness in processing and extracting information from CAD files in the open-source YAML format. These visualizations showcase the versatility and accuracy of the algorithm when applied to various geometrical shapes and models.

1. Efficient data structures-NumPy arrays were utilised to compute normal vectors and cross-products quicker. Using NumPy vectorized operations instead of for-loops enhanced nth-order equation and cross-product computations.

2. Parallel processing-Using multiprocessing or multithreading to generate normal vectors for various surface types improved the whole process greatly. This improved performance and accuracy during CAD translation from STEP to HDF5 files.

3. Code Refactoring-A lot of enhancements have resulted in code reorganisation in method 'convert_data_to_hdf5' The function has been divided into logical sections by establishing independent functions such as 'convert_dict_to_hdf5' and 'convert_stat_to_hdf5' This improves the readability and maintainability of the code. The function now handles various data types, such as dictionaries and lists, uniformly. It analyses dictionaries recursively and generates datasets for numeric data arrays, resulting in cleaner and more organised code.

4. Memory optimisation- The CADConversion algorithm has been managing memory effectively in order to eliminate needless copies or excessive memory utilisation during data handling. Depending on the task ID, the script then fetches the list of files (FILES_TO_COPY) intended for processing. If no files are provided for a certain job, the script terminates gracefully and hence frees the space.

5. Streaming and chunking- The CADConversion algorithm used this technique for rapidly processing huge CAD dataset files by dividing them down into smaller, manageable chunks. This method is especially effective for dealing with datasets that are too huge to fit whole in memory, resulting in memory depletion and lengthy processing times [35]. Rather than loading the complete CAD file into memory, the programme reads a bit of it, processes it, and then goes on to the next section. This procedure is repeated until the full file has been processed. Streaming helps the program to operate with enormous datasets without consuming a lot of memory.

Chapter Seven

Conclusion and Future Work

7.1 Summary of Contributions and Findings

By successfully converting 99 percent of the Fusion360 dataset of 2,56,618 CAD models to open-source and cross-language binary HDF5 format, Our work provides a new resource for the advancement of geometric deep learning, with an emphasis on applications centred on human-created, mechanical designs. Researchers will be able to convert existing huge and realistic collections of complex cad data to a portable open-source format. The data science community will gain a significant resource and a new baseline for assessing new approaches and algorithms- This enables data scientists to access and analyse the data using widely available libraries and tools that support these formats. Data scientists may deal with enormous CAD datasets without facing memory limits by converting the CAD models to HDF5. The open-source YAML and HDF5 formats are platform-agnostic and may be accessed and utilised by a variety of computer languages. This improves interoperability and allows data scientists to work with CAD models in the programming environments of their choice.

7.2 Limitations and Future Work

Ensuring the project's success, there are also certain limitations and areas that require further research work.

1. Releasing CADConversion library on PIP conda : PIP and Conda are popular package managers in the Python and data science communities. The CADConversion library is now available on PIP and Conda, allowing users to simply install and manage it as a Python package. As a result, the library is now ready for use in a variety of Python-based applications, processes, and settings. Users only need to execute a few commands to install the library and begin converting CAD files using the available functions and tools.

2. Documentation: Extensive documentation is required for the CADConversion library to be adopted and used successfully. It should provide extensive explanations of the library's features, use examples, and an API reference. Furthermore, the documentation should provide step-by-step instructions for installing the library, configuring dependencies, and performing typical CAD file conversion operations. Code samples and tutorials that are well described can assist users in rapidly grasping the library's capabilities and efficiently incorporating them into their projects.

3. Implementing CI/CD for future versions: The Continuous Integration and Continuous Deployment pipeline, i.e. CI/CD, must verify that the algorithm is current and trustworthy. CI/CD must allow for quick iteration and creation of new features or upgrades, ensuring that the project stays relevant and usable by the CAD community. The library's versioning guarantees that users may select the best version for their needs, while continuous integration (CI) ensures the library's stability and dependability.

4. Data integration and deployment with PyTorch and TensorFlow: Once the CAD data is in HDF5 format, it can be simply imported into PyTorch and TensorFlow for machine

learning model training. PyTorch and TensorFlow both offer robust libraries and tools for developing, training and assessing machine learning models. Hence we can make use of these frameworks to create complex models that employ CAD data as input to generate predictions, categorise objects, and perform other activities.

5. Cloud-Based Solutions: Using cloud computing platforms like AWS, Azure, or Google Cloud to leverage their scalable architecture and resources will help leverage this conversion algorithm.

7.3 Conclusion and Final Remarks

The proposed future work will enhance and build on the project's accomplishments, ensuring that the established algorithm remains a useful resource for the CAD community and numerous businesses that rely on CAD systems for design and production processes. The project intends to make a substantial contribution to the CAD sector by developing a revolutionary algorithm that effectively converts CAD files in the open-source HDF5 format.

REFERENCES

[1] Gerald Farin, Josef Hoschek, and M-S Kim. *Handbook of computer aided geometric design*. Elsevier, 2002.

[2] Benjamin Marussig and Thomas JR Hughes. "A review of trimming in isogeometric analysis: challenges, data exchange and simulation aspects". In: *Archives of computational methods in engineering* 25 (2018), pp. 1059–1127.

[3] Sang-Hyun Park. Sungmin Kim. "Conversion from CAD Models to B-Rep Models for Automatic Finite Element Analysis". In: (2016).

[4] MJ Pratt. "Introduction to ISO 10303—the STEP standard for product data exchange". In: (2001).

[5] *The official Fusion360Gallery Dataset web site*. URL: https://github.com/AutodeskAILab/Fusion360GalleryDataset.

[6] Sebastian Koch et al. "ABC: A Big CAD Model Dataset for Geometric Deep Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[7] Zhirong Wu et al. "3D ShapeNets: A Deep Representation for Volumetric Shapes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015.

[8] Salvatore Gerbino. "TOOLS FOR THE INTEROPERABILITY AMONG CAD SYSTEMS". In: (Sept. 2013).

[9] Mark Palmer Stephen Cawood. "STEP Interoperability - A comparison of available CAD formats to ISO 10303". In: (2013).

[10] Christian Feldmann Andreas Egger and Peter Wriggers. "Towards a data model for integrated CAD and analysis using STEP and HDF5"". In: (2022).

[11] Xi Vincent Wang and Xun W Xu. "A collaborative product data exchange environment based on STEP". In: *International Journal of Computer Integrated Manufacturing* 28.1 (2015), pp. 75–86.

[12] Lars Mescheder et al. "Occupancy networks: Learning 3d reconstruction in function space". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4460–4470.

[13] Abhishek Sharma and Maks Ovsjanikov. "Weakly supervised deep functional maps for shape matching". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 19264–19275.

[14] Isabelle Pfander, Hailey Johnson, and Sean Arms. "Comparing read times of zarr, HDF5 and netCDF data formats". In: *AGU fall meeting abstracts*. Vol. 2021. 2021, IN15A–08.

[15] *The official h5py Library web site*. URL: https://www.h5py.org/.

[16] *The official meshio Library web site*. URL: https://pypi.org/project/meshio/2.3.5/.

[17] Wenming Cao et al. "A comprehensive survey on geometric deep learning". In: *IEEE Access* 8 (2020), pp. 35929–35949.

[18] I Kolingerová. "Computational geometry education for computer graphics students". In: (2008).

[19] Jarek Rossignac. "Beyond solid modelling". eng. In: *Computer aided design* 23.1 (1991), pp. 2–3. ISSN: 0010-4485.

[20] Lovett Banchoff T.F. "Differential Geometry of Curves and Surfaces". In: (2010).

[21] Tamas Varady and R. Martin. "The Handbook of Computer-Aided Geometric Design". In: Jan. 2002, pp. 651–681.

[22] *Handbook of computer aided geometric design*. eng. 1st ed. Amsterdam ; Elsevier, 2002. ISBN: 1-281-03630-7.

[23] *The official YAML web site*. URL: https://yaml.org/.

[24] Karl DD Willis et al. "JoinABLe: Learning Bottom-up Assembly of Parametric CAD Joints". In: *arXiv preprint arXiv:2111.12772* (2021).

[25] Karl D. D. Willis et al. "Fusion 360 Gallery: A Dataset and Environment for Programmatic CAD Construction from Human Design Sequences". In: *ACM Transactions on Graphics (TOG)* 40.4 (2021).

[26] Joseph G. Lambourne et al. "BRepNet: A Topological Message Passing System for Solid Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 12773–12782.

[27] Onur Rauf Bingol and Adarsh Krishnamurthy. "NURBS-Python: An open-source object-oriented NURBS modeling framework in Python". In: *SoftwareX* 9 (2019), pp. 85–94.

[28] *The official Compute Canada web site*. URL: https://osf.io/k89fh/wiki/Compute%20Canada/.

[29] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[30] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[31] Aaron Meurer et al. "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. URL: https://doi.org/10.7717/peerj-cs.103.

[32] skoch9. *Skoch9/meshplot: Plot 3D triangle meshes*. URL: https://github.com/skoch9/meshplot.

[33] Infinidat. *Munch*. 2023. URL: https://github.com/Infinidat/munch.

[34] Vamsi Naga Sai Chandra Madduri. "SmartCAD For Exploring Complex CAD files in YAML Format". In: (2023).

[35] de Berg Mark et al. *Computational geometry algorithms and applications*. Spinger, 2008.