SmartCAD

For

Exploring Complex CAD files in YAML Format

by

Vamsi Naga Sai Chandra Madduri
B.E., Sathyabama Institute of Science and Technology, 2018

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

MASTER OF SCIENCE

in the Department of Computer Science

SmartCAD

For

Exploring Complex CAD files in YAML Format

by

Vamsi Naga Sai Chandra Madduri
B.E., Sathyabama Institute of Science and Technology, 2018

Supervisory Committee

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

Dr. Jens Weber, Departmental Member
(Department of Computer Science)

ABSTRACT

In this study, we develop an efficient approach for sampling and extracting data from Computer-Aided Design And Manufacturing (CAD/CAM) files using an open-source YAML format. Our project is driven by the challenges encountered while performing file conversions between multiple CAD software and the complex data organization within CAD file structures, including proprietary, Model-Based Definition (MBD), and non-MBD CAD files. By employing the YAML format for CAD files, our approach eliminates the need for third-party software for file conversions, reduces compatibility issues, and improves the efficiency of CAD data management and processing.

To achieve this, we implement a methodology involving preprocessing, feature extraction, and computing winding numbers from the YAML-formatted CAD files. The proposed method serves as a foundation for a future project that will convert approximately one million CAD models into the YAML format and leverage the extracted data for various purposes, such as model learning and data analysis. The findings of this study contribute to a deeper understanding of CAD data processing techniques and provide a foundation for further research and development. Our work addresses the current limitations of CAD systems, including the restrictions of conventional neutral 3D CAD file formats. It paves the way for more effective and streamlined workflows in industries that rely heavily on CAD technology.

Contents

List of Figures

ACKNOWLEDGMENTS

Firstly, I want to express my whole-hearted gratitude to Dr.Teseo Schneider for his constant support and guidance throughout this project. His dedication and mentorship throughout my study at University of Victoria are meritorious.

The University of Victoria and the Department of Computer Science have played a significant role by creating an excellent environment for me to improve and hone my technical and employable skills.

I am incredibly thankful to my dear friends Neha Koulecar, Priyadharsini Srinivasan, and Bhanu Sree Tirumalasetti for their unwavering support and encouragement. They have indeed been the pillars of support during my most challenging times.

My teammate, Sachin Chaudhary's efforts and contributions to this project are commendable. I am truly fortunate to have had the opportunity to work alongside such a dedicated and talented individual.

DEDICATION

I dedicate this project to my loving family, To my Mom, Krishna Veni Madduri, who has always believed in me, supported, and encouraged me throughout my academic journey. To my Father, Srinivasarao Madduri, whose guidance and wisdom have shaped me into the person I am today, I offer my heartfelt thanks to my brother, Harish Vardhan Madduri, who has been my confidant and friend. Thank you for always being there for me and making me believe in myself.

Chapter One

Introduction

1.1 Background and Motivation

The development and usage of computer-aided design (CAD) systems have transformed how various industries, such as manufacturing, automotive, and aerospace, design and create their products [1]. However, the CAD landscape is fragmented, with multiple CAD proprietary software, file formats and versions. Some popular CAD software programs and their formats include Autodesk AutoCAD (DWG and DXF), Dassault Systèmes CATIA (CATPart and CATProduct), PTC Creo Parametric (PRT and ASM), Siemens NX (PRT), Dassault Systèmes SOLIDWORKS (SLDPRT, SLDASM, and SLDDRW), Autodesk Inventor (IPT, IAM, and IDW), Bentley Systems MicroStation (DGN), and Bricsys BricsCAD (DWG and DXF) [2]. This fragmentation leads to challenges in converting one CAD file version to another, with each requiring a third-party software dependency to function.

To address these challenges, some efforts have been made to create universal or neutral file formats, such as the Standard for the Exchange of Product Model Data (STEP) [3]. Another issue is the complex file structure and data organization in CAD file formats, which heavily rely on CAD kernels to interpret and manipulate the information [4, 5]. These complexities hinder the interoperability and flexibility of CAD systems, necessitating an alternative solution.

1.2 Contribution and Significance of the Study

Given the challenges mentioned above, we at the University of Victoria have proposed and developed an effective algorithm to effectively sample and extract the CAD data stored information into an open-source format. The project's primary goal is to enhance the accessibility and versatility of CAD information across different industries and applications by providing an alternative approach for processing and analyzing CAD data. With the data accessibility and the use of open source format of CAD models this projects aims to eliminates the need for third-party software usage for file conversions. It also reduces compatibility issues, significantly improving CAD data handling and processing efficiency. Additionally, the algorithm can be applied to a large dataset of CAD files, such as the ABC dataset with 1 million CAD models. This makes it a valuable resource for data science benchmarking and analysis, as previously demonstrated in studies by Koch et al. [6] and Willis et al. [7].

1.3 Ongoing Project: Converting CAD Models to YAML Format

In another ongoing project, our team is diligently working on converting approximately one million CAD models into the YAML format. The primary focus of this project lies in the sampling and feature extraction of the converted YAML files. The extracted features and samples can be effectively utilized for various purposes, such as model learning and data analysis. By harnessing this valuable information, we aim to enhance the overall understanding and performance of our analytical processes and machine learning models.

1.4 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 processing, geometry modeling, and feature extraction.
- Chapter Three introduces the terminology, definitions, and technical background required for understanding the proposed algorithm and its implementation.
- Chapter Four details the design of the proposed algorithm and the system architecture that supports the sampling and extraction of information from CAD files in the open-source YAML format.
- Chapter Five discusses the implementation of the proposed algorithm, including the programming language and tools used, as well as the specific components of the implementation.
- Chapter Six presents the testing, validation, optimization, and analysis of the algorithm's performance 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 analysis.

Chapter Two

Background Work

2.1 Literature Review

2.1.1 Geometric Modeling

In the field of geometric modeling, Stroud et al. [8] provide a comprehensive review, delving into various representations, operations, and applications. They emphasize the significance of representations such as boundary representation (B-rep) and constructive solid geometry (CSG). Requicha [9] initially introduced B-rep and CSG, which Requicha & Voelcker [4] later developed. These representations play a crucial role in understanding and manipulating complex geometries. Furthermore, Stroud et al. discuss parametric curves and surfaces, initially proposed by Sederberg et al. [10]. Parametric representations enable more efficient and accurate modeling of complex shapes. The authors also explore implicit representations, a topic extensively covered by Shapiro [5], which have a significant impact on the development of computer-aided design (CAD) systems, as outlined by Zeid [1].

2.1.2 3D Curve and Surface Processing

In recent years, machine learning and data-driven approaches have made significant strides in the field of 3D curve and surface processing, encompassing generation, segmentation,

reconstruction, and registration. These advancements offer new opportunities for shape analysis, synthesis, and manipulation. One study to cite as an example would be of Groueix et al. [11], which employs deep learning techniques, namely Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) for generating realistic 3D shapes by creating a collection of parametric surface elements (atlases).

Similarly, another study by the same author mentions about surface segmentation., Groueix et al. [12] utilized unsupervised learning techniques to segment 3D shapes into meaningful parts, building upon traditional methods like spectral clustering and region growing. In the domain of surface registration, Litany et al. [13] developed Deep Geometric Functional Maps to find correspondences and merge multiple scans into a single model. Their approach combines traditional techniques, such as Iterative Closest Point (ICP), with geometric deep learning, allowing for the handling of non-rigid deformations and large-scale point clouds.

Finally, for surface reconstruction, Mescheder et al. [14] introduced Occupancy Networks, which employ continuous volumetric functions to represent 3D surfaces. By leveraging techniques like learning implicit representations or occupancy functions, this approach demonstrates the potential of neural network-based methods in reconstructing 3D surfaces from geometric data.

2.1.3 Mesh Repair

Mesh repair is another critical aspect of geometry processing. Attene [15] proposes a lightweight approach to repairing digitized polygon meshes, contributing to the improvement of the quality and usability of the extracted geometry data. This method can help overcome issues such as holes, degenerate faces, and self-intersections in the generated mesh models, which may hinder proper analysis and manipulation of the geometries.

2.1.4 Point Cloud Denoising

Zhang and Wang [16] present a method for point cloud denoising using 3D dictionary learning. This approach can be instrumental in refining noisy data obtained from CAD files. Reducing the noise in the point cloud data makes it possible to extract more accurate and reliable geometric features, enhancing the performance of subsequent geometry and topology analysis tasks.

2.2 Related Tools and Libraries

2.2.1 Geometry Processing Libraries

Geometry processing libraries have revolutionized the field of geometry processing. A software project called CGAL [17] offers simple access to trustworthy and effective geometric algorithms in the form of a C++ library. Geographic information systems, computer-aided design, molecular biology, medical imaging, computer graphics, and robotics are just a few fields that use CGAL to perform geometric computations.

The library provides a variety of data structures and algorithms, including triangulations, Voronoi diagrams, Boolean operations on polygons and polyhedra, point set processing, curve arrangements, surface and volume mesh generation, geometry processing, alpha shapes, convex hull algorithms, shape reconstruction, AABB, and KD trees. libigl [18], is another simple C++ geometry processing library that offers various functionalities for processing and analyzing geometric models.

2.2.2 Mesh Processing Tools

MeshLab is an open-source and user-friendly interface used manipuation and processing of complex mesh structures. It supports extensive processing functions as for cleaning, repairing, simplifying the mesh models. It was introduced by Cignoni et al [19]

2.2.3 Challenges and Opportunities in Geometry Processing

This literature review provides a comprehensive foundation for the project; however, there are several challenges and opportunities in the field of geometry processing that warrant further research.

Geometric deep learning [20] refers to the application of deep learning techniques to problems that involve non-Euclidean data, such as 3D shapes and graphs. These models have shown great potential in various tasks, but they often face scalability challenges, requiring large amounts of data for training.

One of the issues with training deep geometric models in the past was the unstructured nature of CAD data. However, with the conversion of CAD data to a structured format in this project, this problem can be eliminated. This development paves the way for future research to effectively train deep geometric models using the structured data provided by our platform, thus expanding the potential applications of deep learning in the field of geometry processing.

2.2.4 Project Overview and Connection to Background Work

The current project focuses on developing and applying various algorithms to effectively sample, extract, and verify CAD data in an open-source format. This effort aims to enhance the accessibility and versatility of CAD information across different industries and applications, addressing the challenges of fragmentation, interoperability, and flexibility in CAD systems. The ultimate goal is to improve CAD data handling and processing efficiency without creating new datasets in the current project.

In anticipation of future endeavors, our team plans to work on a project that will utilize the findings and methods from the current project for sampling and data extraction. This future project will involve converting approximately one million CAD models into the YAML format and experimenting with the extracted data for various purposes, such as model

learning and data analysis. By harnessing the valuable information from both the current and future projects, we aim to enhance the overall understanding and performance of our analytical processes and machine learning models.

The Background Work chapter offers a thorough foundation for both projects by reviewing relevant literature and addressing challenges in geometry processing. This knowledge helps situate the projects within the broader context and emphasizes their potential contributions to the field.

Chapter Three

Preliminaries

The background knowledge and key terminologies that aid in further understanding the project is mentioned in the following sections. Section 3.1 lists the glossary of key terms and their meanings used in this report. The Section 3.2 covers the technical background, including mathematical and computational concepts and data structures relevant to this project.

3.1 Definitions and Terminology

- Geometry [21]: Geometry is a branch of mathematics that deals with the properties of geometric shapes such as lines, curves, points, surfaces and studies their relations and measurements,etc. This field of geometry has varied applications in computer graphics, engineering and physics.

- Topology [21]: It deals with Studying the properties of the above-mentioned geometric objects which do not undergo any form of deformation and constantly remain invariable. Topology to as rubber sheet geometry. Topology is used in multiple sub-topics of mathematics such as " dynamic systems, differential equations, knot theory, Riemann surfaces in complex analysis" along with string theory in physics and in studying space-time structures. [22]

- Bounding Box [23]: The smallest enclosing box that contains a geometric object, usually represented by minimum and maximum coordinates for each axis. Bounding boxes simplify and speed up geometric computations, such as collision detection and rendering.

- Solids [24]: Enclosed 3D objects with a defined finite volume. They are the primary objects of interest in geometric modelling and are constructed using B-rep (Boundary representation), CSG (constructive solid geometry), and volumetric representations.

- 2D/3D Curves [25]: Two-dimensional and Three-dimensional curves which can be defined using geometric equations and specific coordinates in space. Examples include lines, circles, Bezier curves, helics and B-spline curves

- Vertices [26]: Defined as points in space, which are identified by their coordinates, that contribute to the shape and size of geometric objects. Vertices serve as fundamental units of these shapes and are used to represent them in computer graphics and CAD systems. Below image 3.1 contains an example of Vertex.

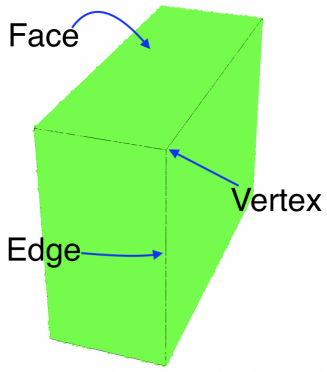

Figure 3.1 Vertex, Face and Edge of a shape(BOX)

- Edges [27]: Edges are Line segments or curves connecting vertices in a geometric object and defining the boundaries of faces. They represent these connections and structures

in topological representations such as boundaries (B-rep). Example Fig 3.1.

- Faces [8]: Flat or curved surfaces constitute a 3D geometric object, known as faces. These faces are defined by their bounding loops, which determine their shape. They are employed to represent external and internal boundaries of objects in boundary representation (B-rep) systems. An example shown in Example Fig 3.1

- Halfedges [28]: Directed edges that store topology information such as object orientation and adjacency; used in data structures like DCEL (doubly connected edge list) to efficiently represent and manipulate the object's topology.

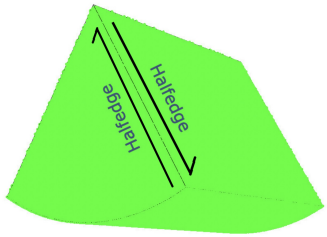

Figure 3.2 Half-edge of a shape(Cylinder Wedge)

- Loops [26]: Sequences of halfedges that define the boundary of a face. Loops represent the connectivity between edges and faces in boundary representation (B-rep) systems.

- Shells [27]: Collections of faces together form a closed boundary around a solid or a hole in a solid. Shells represent objects' outer and inner boundaries in boundary representation (B-rep) systems.

3.2 Technical Background

The information required to further understand this project along with the technicalities such as winding numbers, the geomdl package, NURBS, and B-splines. Further references for reading are attached.

3.2.1 Computational Geometry

It studies algorithms and data structures for solving geometry-related problems. Some examples include Convex hulls, Voronoi diagrams, and mesh generation. In this project, computational geometry is crucial for manipulating and analyzing geometric objects and their relationships [23, 26].

3.2.2 Winding Numbers

Winding numbers are used in computational geometry to determine if a point lies inside or outside a polygon [29]. The winding number of a point concerning a closed curve is the total number of times the curve wraps around the point in a counterclockwise direction. In 2D geometry, winding numbers determine whether a point lies inside a polygon by computing the sum of the angles between the point and each pair of adjacent vertices in the polygon. If the winding number is nonzero, the point lies inside the polygon; if it is zero, it lies outside the polygon.

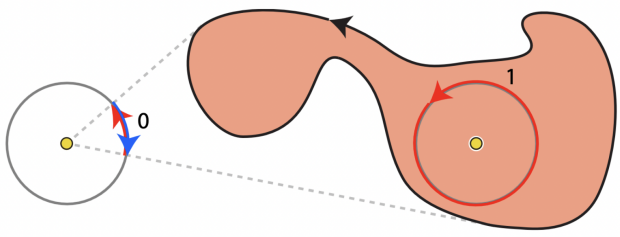

Figure 4: *Winding number is the signed length of the projection of a curve onto a circle at a given a point divided by* 2π*. Outside the curve, the projection cancels itself out. Inside, it measures one.*

Figure 3.3 Winding Number explanation from [29]

3.2.3 Geomdl Package

The geomdl package [30] is a Python library for geometric modelling that supports various curve and surface types, including NURBS (Non-Uniform Rational B-splines) and B-splines.

It provides tools for creating, evaluating, and manipulating curves and surfaces and importing and exporting geometry data in different formats. The package is well-suited for projects that involve geometric modelling and can be easily integrated with other Python libraries for visualization and analysis.

3.2.4 NURBS

Non-Uniform Rational B-splines (NURBS) [31] are mathematical representations of 3D geometry that can accurately model any shape, from simple 2D shapes like lines and circles to complex 3D free-form surfaces. NURBS are widely used in computer graphics and computer-aided design (CAD) due to their flexibility and precision. NURBS curves and surfaces are defined by control points and associated weights, which determine their shape and smoothness.

3.2.5 B-splines

B-splines [32] are piecewise polynomial functions used in geometric modelling to represent curves and surfaces. They are defined by a set of control points and degrees, determining the curve's or surface's smoothness. B-splines can be uniform, meaning the knots (the points where the individual polynomial pieces join) are evenly spaced, or non-uniform, meaning the knots can be placed arbitrarily. NURBS can be considered a generalization of B-splines, as they incorporate weights to provide more control over the shape of the curve or surface.

Chapter Four

Algorithm Design and System Architecture

4.1 INTRODUCTION

This chapter aims to provide a comprehensive overview of the algorithm design process used in the SmartCad algorithm. This chapter will focus on the problem statement, data, system architecture, proposed solution and design, while the previous chapter provides the necessary background knowledge.

4.2 YAML CAD Data

The geometry and topology files define the geometric and topological information, as shown in figures 4.1 and 4.2.

Geometry Data: The geometry file provides information about the shape, size, and location of the part. It includes data about 2D curves, 3D curves, bounding box, surfaces, and vertices. Specifically, it defines a 2D curve with specific type defined for each curve, shape, location, radius, and orientation (x_axis, y_axis). Similarly, it represents a 3D curve with the same properties plus a z_axis. A bounding box is provided to define the limits

of the shape in 3D space. The surface is described with coefficients, location, trim domain, and orientation (x_axis, y_axis, z_axis) —finally, vertices for the geometry. The surface patches include Plane, Cylinder, Cone, Sphere, Torus, Revolution, Extrusion, BSpline, and Others. The various curve types include Line, Circle, Ellipse, BSpline, and Other.

Topology Data: The topology file describes the structure and relationships between the various shape elements. It contains information about edges, faces, half edges, loops, shells, and solids. Edges are defined with reference to a 3D curve, start vertex, and end vertex. Faces are described with reference to the surface and loops. The half-edges section references a 2D curve, an edge, and orientation information. Loops and shells are defined with references to the corresponding half-edges and faces, respectively.

In figure 4.1, The geometry file describes a circle in both 2D and 3D space. The file is structured as follows:

- parts: Contains the main geometrical components.
- 2dcurves: Contains a list of 2D curves.
  - interval: Parameter range for the curve. In this case, it ranges from 0 to 2π (6.283185307179586).
  - location: Center point of the circle in 2D space (x, y).
  - radius: Radius of the circle (39.3715863766753).
  - type: Type of curve, which is a Circle.
  - x_axis: The x-axis direction vector.
  - y_axis: The y-axis direction vector.
- 3dcurves: Contains a list of 3D curves.
  - interval: Parameter range for the curve. In this case, it ranges from 0 to 2π (6.283185307179586).

  - location: Center point of the circle in 3D space (x, y, z).
  - radius: Radius of the circle (39.3715863766753).
  - type: Type of curve, which is a Circle.
  - x_axis: The x-axis direction vector.
  - y_axis: The y-axis direction vector.
  - z_axis: The z-axis direction vector.

- bbox: The bounding box of the geometry with minimum and maximum coordinates for x, y, and z.
- surfaces: Contains a list of surfaces.
  - coefficients: Coefficients for the surface equation.
  - location: Reference point on the surface.
  - trim_domain: The trimming domain for the surface.
  - type: Type of surface, which is a Plane.
  - x_axis: The x-axis direction vector.
  - y_axis: The y-axis direction vector.
  - z_axis: The z-axis direction vector.
- vertices: List of vertices for the geometry. In this case, only one vertex exists (39.3715863766753, 0.0, 0.0).
- version: The version of the file format ('2.01').

In figure 4.2, The topology file represents the topological structure of a circle. The file is structured as follows:

- parts: Contains the main topological components.

- edges: Contains a list of edges.
  - 3dcurve: Index of the corresponding 3D curve in the geometry file.
  - end_vertex: Index of the ending vertex for the edge.
  - start_vertex: Index of the starting vertex for the edge.
- faces: Contains a list of faces.
  - loops: List of loop indices that define the face boundary.
  - surface: Index of the corresponding surface in the geometry file.
  - surface_orientation: Indicates whether the face orientation is consistent with the surface's orientation.
- half-edges: Contains a list of half-edges.
  - 2dcurve: Index of the corresponding 2D curve in the geometry file.
  - edge: Index of the associated edge.
  - mates: List of indices of half-edge mates.
  - orientation_wrt_edge: Indicates whether the half-edge orientation is consistent with the edge's orientation.
- loops: Contains a list of loops.

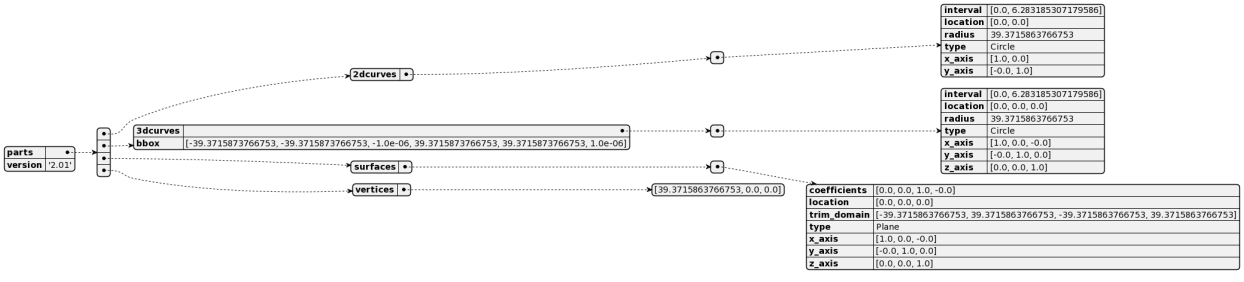

Figure 4.1 A sample Geometry data of shape Circle

  - half-edges: List of half-edge indices that form the loop.

- shells: Contains a list of shells.
  - faces: List of face entries with face indices and orientation with respect to the shell.
  - face_index: Index of the face.
  - face_orientation_wrt_shell: Indicates whether the face orientation is consistent with the shell's orientation.
  - orientation_wrt_solid: Indicates whether the shell's orientation is consistent with the solid's orientation.

- solids: An empty list in this case, as there are no solids for a circle.

- version: The version of the file format ('2.01').

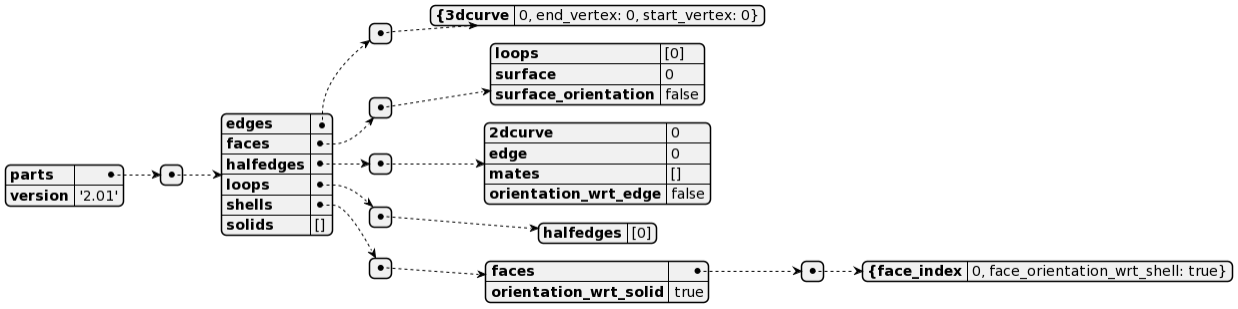

Figure 4.2 A sample Topology data of shape Circle

Together, these files provide a complete description of a circle, including its geometric properties and the relationships between its elements, enabling any further analysis or manipulation of the shape.

4.3 System Architecture

This section presents the system architecture of a smart CAD application designed to process and analyze information about parts, topologies, and geometries. As shown in Fig. 4.3, the UML class diagram provides an overview of the application, encompassing seven classes, each serving a specific purpose.

The initial class, PartDataProcessor, is responsible for part-related data processing. It has two private attributes: part_topology and part_geometry, both of the types PartTopology and PartGeometry. A public method called process_part_data() is present within this

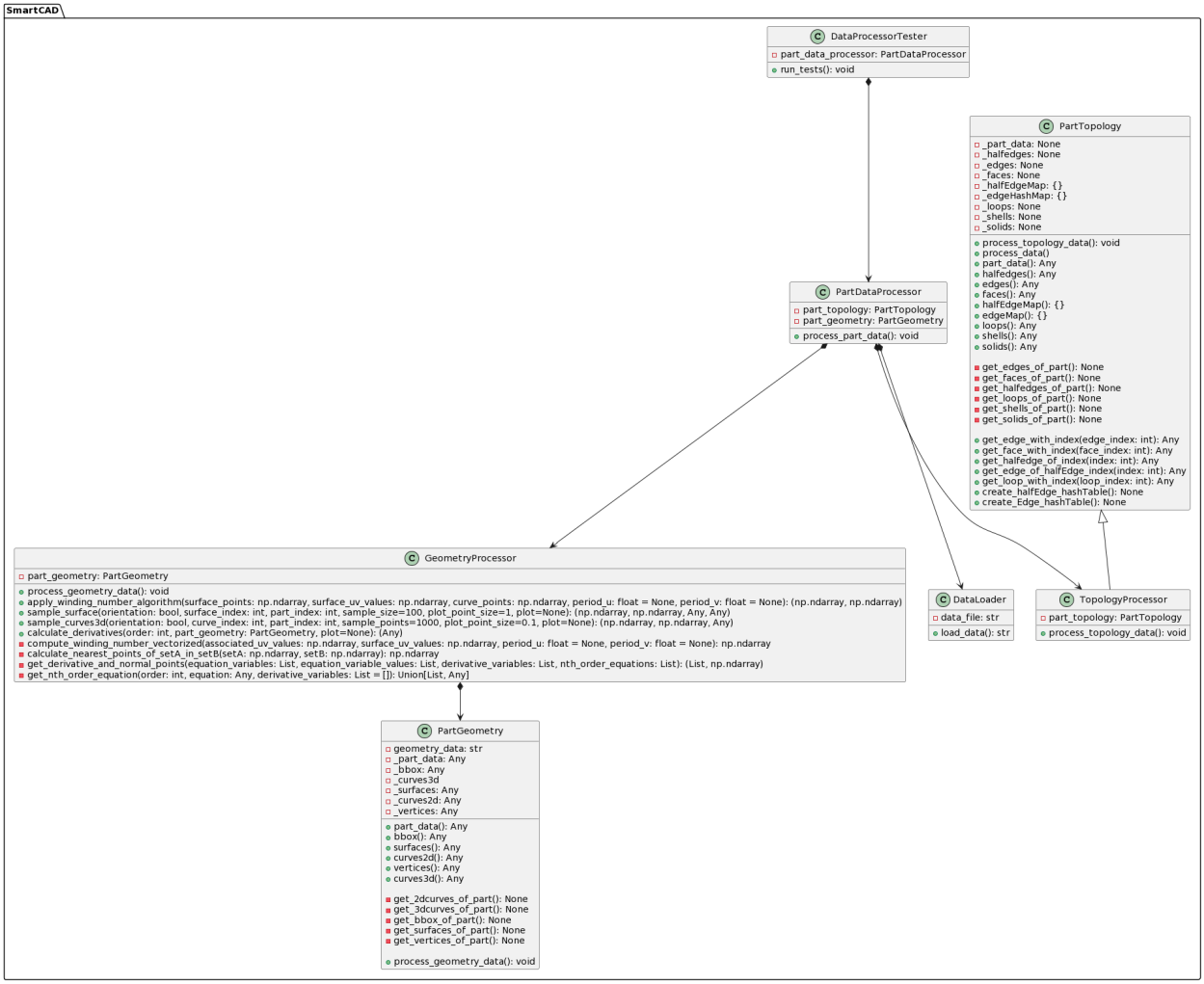

Figure 4.3 Class diagram of SmartCAD Project

class, requiring no arguments and returning void.

The second class, TopologyProcessor, focuses on processing topology data and has a private attribute, part_topology, of the type PartTopology. It contains a public method called process_topology_data() which takes no arguments and returns void.

The third class, GeometryProcessor, is dedicated to processing geometry data and includes a private attribute, part_geometry, of the type PartGeometry. The GeometryProcessor class contains multiple public methods to perform various geometric calculations.

The PartTopology class, which extends the TopologyProcessor class, represents the topology of a part and contains numerous private attributes and methods dedicated to topology data processing. Similarly, the PartGeometry class extends the GeometryProcessor class and deals with the geometry of a part, featuring several private attributes and methods for geometry data processing.

The sixth class, DataLoader, is responsible for loading data and includes a private attribute, data_file, of type string. DataLoader has a public method, load_data(), which returns a string. The final class, DataProcessorTester, serves as a tester for the data processor and contains a private attribute, part_data_processor, of the type PartDataProcessor class. It has a public method, run_tests(), which takes no arguments and returns void.

To summarize the UML class diagram provides a precise view of the smart CAD application and its object oriented design which processes and analyses data about topologies, parts and geometries. It also emphasizes the relationship among classes, methods and attributes and provide a application implementation framework. A detailed explanation of the code with snippets will be provided in the subsequent chapter.

4.4 Proposed Solution and Design

The proposed solution involves designing and implementing a CAD data processing algorithm capable of extracting useful geometric information from complex CAD models. This algorithm is based on a combination of geometry and topology processing techniques and efficient data structures for storing and manipulating CAD data. The algorithm's logic and calculations are as follows:

- Pre-processing: Validate the input CAD data and initialize the required data structures, such as PartGeometry and TopologyProcessor objects, to store and process the data.

- Data extraction and linking: PartGeometry class extracts and stores the required geometric information, such as surfaces, 2D curves, 3D curves, vertices, and bounding boxes, from the CAD data. PartTopology class is used for topology data extraction and formation of a half-edge table; PartDataProcessor class uses this information for further sampling and data extraction

- Shape sampling: sample_all_shapes() is the method used for iterating the CAD data hierarchy such as solids, loops, faces parts, etc. And invokes the sample_surface(), sample_curve3d(), and apply_winding_number_algorithm() methods for processing the surfaces and curves using winding number algorithm

- Winding number algorithm: This algorithm determines the interior and exterior regions of the CAD model which thereby aids in feature identification such as cavity or holes in the model.

- Visualization and storage: Followed by the processing of all the shapes is the visualization of the resulting geometry using color shades that represent these geometrical surfaces and curves. The generated visualizations are saved with a naming convention that reflects the part index., and extracted data can be used for learning purposes.

Several assumptions are made to simplify the implementation throughout the algorithm design process, such as assuming the input converted YAML data of CAD Models is well-formed and adheres to a specific structure. These assumptions can be relaxed or adjusted as needed to accommodate different CAD data formats or variations in the data.

Chapter Five

Algorithm Implementation

This chapter provides the detailed implementation of the SMARTCAD data processing algorithm. The chapter is organized into four sections. Section 5.1 covers programming language, libraries and frameworks used in the implementation process and rationale behind the choice, Section 5.2 covers detailed explanation of the implementation process including code snippets and pseudocode. In next chapter we covered testing process used to validate the algorithm's accuracy and efficiency, the process of optimizing the algorithm's performance and Results.

5.1 Programming Language and Tools

We choose the Python[33] programming language for the project implementation due to its readability, rich library support, and widespread acceptance in the scientific and engineering communities. Python has a vast ecosystem of tools in scientific computing that enable faster implementation and efficient code execution.

The following libraries and frameworks were used in the implementation process:

1. NumPy: For numerical computing and efficient array manipulation, it provides a high-performance array object and functions for working with these arrays. [34]

2. SciPy: For scientific computing and mathematical functions, offering modules for optimization, linear algebra, integration, and interpolation, etc. [35]

3. geomdl - a library used for geometric modeling and computer-aided design (CAD) applications, specifically for working with B-spline and NURBS curves and surfaces. [30]

4. Meshplot is a Python library for the 3D visualization of mesh data. It provides an efficient and easy-to-use interface for plotting and rendering 3D models. [36]

5. Sympy: This library is used to manipulate mathematical expressions symbolically, perform calculus, algebraic operations, and solve equations. [37]

6. PyYAML (short for "YAML Ain't Markup Language")[38] Python library allows for easy parsing and generating YAML files in Python code.

7. Munch: Python library that provides a dictionary-like object (called a "Munch") with attribute-style access to its keys. [39]

5.2 Implementation Details

The code is well-organized into modules and classes, making it easier to update and maintain. The main modules include:

1. Parsing CAD data and extracting topological information: The Topology Processor reads the CAD file and extracts topological information such as parts, solids, shells, faces, loops, and edges. This information is stored in appropriate data structures to facilitate further processing.

2. Processing geometric information: The Geometry Processor processes the geometric information associated with the topological elements. This includes sampling surfaces and curves, computing the winding number for points on the surfaces, and generating visualizations of the processed data.

3. Generating output: The processed data are saved as images or other appropriate output formats for further analysis or visualization.

All the modules are further divided into seven classes. We will go through the essential definitions of each class below.

5.2.1 CAD Data Loading

The DataLoader class is responsible for loading data from a given geometry model file. This class has a single method called load_yaml_file(file_path) that takes the file path as an input argument. The method checks whether the given file path exists and whether the file has a ".yaml" extension. If the file is a valid YAML file, the method reads its contents and returns the data as a Python dictionary using the yaml.load() function from the yaml library. The method raises a ValueError if the file is not a valid YAML file[cite yaml].

Here's the load_yaml_file method implementation:

```
def load_yaml_file(file_path):
    assert Path(file_path).exists(), "Please provide valid file path"
    # Check if file is yaml file
    if Path(file_path).suffix == ".yaml":
        with open(file_path, "r") as fp:
            return yaml.load(fp, Loader=CLoader)
    else:
        raise ValueError("Please provide valid yaml file")
```

Additionally, the Munch.fromDict() function is used to convert the dictionary obtained from the YAML file into an object-like structure, making it easier to access the data using dot notation. This function is part of the Munch library, which is a wrapper around Python dictionaries for attribute-style access.

Example usage:

```
data = DataLoader.load_yaml_file("file_path.yaml")

data_munch = Munch.fromDict(data)
```

5.2.2 Data Processing and Organization

PartDataProcessor class manages the processing of CAD part data using the GeometryProcessor and TopologyProcessor classes. It contains several methods that manage the processing of geometry and topology data. The following code snippets and pseudocode explain the implementation of each method in detail.

process_data(self)

This method processes geometry and topology data by initializing the GeometryProcessor and TopologyProcessor objects with the respective file paths. Then, it calls their respective processing methods: process_geometry_data() and process_topo_data().

```
    def process_data(self):
        self._geometry_processor = GeometryProcessor(self.geometry_file_path)
        self._geometry_processor.process_geometry_data()
        self._topology_processor = TopologyProcessor(self.topology_file_path)
        self._topology_processor.process_topo_data()
```

sample_all_shapes(self)

The sample_all_shapes method is designed to process and sample all shapes in the CAD data. It iterates through parts, solids, shells, faces, and loops within the CAD model. This method then initiates a variable plot for each of the part and iterates over the solids and their shells. After which it processes each face within a shell, by sampling its surface points,

UV values, and surface patches through ample_surface() method. The calculation happens in the U-V directions on the bases of the surface types. Upon that is the iteration over each loop in the face by processing halfedges and associated curves. It then samples the 3D curves using the sample_curve3d() method and applies the winding number algorithm with the apply_winding_number_algorithm() method. Finally, the method selects surface points with total winding numbers greater than 0.5 and adds them to the plot. The plot is then saved as an image file with a name based on the part index.

Algorithm 1 Sample all Shapes

```
for each part in topology_processor.parts list do
    for each solid in the current part do
        for each shell in the solid do
            for each face in the shell do
                Sample surface based on type and parameters
                save surface_points, surface_uv_values, and surface_patch variables
                Compute period_u and period_v from surface_patch.trim_domain
                for each loop in the face do
                    for each halfedge in the loop do
                        Get curve indices, orientation, edge, and vertices
                        Sample the 3D curve, save curve_points
                        Apply winding number algorithm, save winding_number_list
                        Add winding_number_list to total_winding_number_list
                    end for
                end for
            end for
        end for
```

18: Get surface values with non-zero winding numbers, plot them

19: Save plot as PNG image and store values

20: **end for**

21: **end for**

5.2.3 Topology Analysis

The TopologyProcessor class is responsible for processing topology of YAML CAD data from a given file path, and it initializes the PartTopology objects. The process_topo_data method processes the topology data by iterating through each part and creating a PartTopology object. It then processes the data for each part and appends it to the _part_objects list.

5.2.4 Topological Representation

The PartTopology class is responsible for processing and managing data of a part by initializing and handling various properties such as halfedges, edges, faces, loops, shells, and solids. It does this through the process_data method, which extracts these properties from the part data and constructs two hash tables for associating halfedges with loops and edges with halfedges using create_halfEdge_hashtable and create_Edge_hashtable methods.

Additionally, the class offers helper methods to retrieve specific instances of edges, faces, halfedges, loops, shells, and solids using their indices. The create_halfEdge_hashtable method generates a hash table connecting halfedges to loops, while the create_Edge_hashtable method establishes a hash table linking edges to halfedges.

The following is the code for creating halfEdge hash table:

```
def create_halfEdge_hashTable(self):
    halfEgdeMap = {}
    loops = self.get_loops_of_part()
    for loop_index, loop in enumerate(loops):
        halfedgeIds = loop.halfedges
```

```
        for halfedgeId in halfedgeIds:
            halfEdgeMapValue: dict = halfEgdeMap.get(halfedgeId, {'loops':
                set()})
            loopsData = halfEdgeMapValue['loops']
            loopsData.add(loop_index)
            halfEdgeMapValue['loops'] = loopsData
            halfEgdeMap[halfedgeId] = halfEdgeMapValue
    return halfEgdeMap
```

The following is the code for creating Edge hash table:

```
def create_Edge_hashTable(self):
    egdeMap = {}
    halfedges = self.get_halfedges_of_part()
    for halfEdge_index, halfEdge in enumerate(halfedges):
        edgeId = halfEdge.edge
        eddgeMapValue: dict = egdeMap.get(edgeId, {'halfEdge': set()})
        haldEdgeIdsData = eddgeMapValue['halfEdge']
        haldEdgeIdsData.add(halfEdge_index)
        eddgeMapValue['halfEdge'] = haldEdgeIdsData
        egdeMap[edgeId] = eddgeMapValue
    return egdeMap
```

5.2.5 Geometric Representation

The PartGeometry class manages the geometry data of a CAD part. It initilizes and offer access to different properties, such as the bounding box, 3D curves, surfaces, 2D curves, and vertices, based on the provided part data. This is achieved through the process_geometry_data method, which extracts the necessary information from the part data. However, if the part

data is invalid, the method raises a ValueError.

5.2.6 Geometry Processing and Feature Extraction

The GeometryProcessor class works in conjunction with the PartGeometry class we provided earlier, which processes the individual parts' geometry data. The GeometryProcessor class processes the geometry data for all parts in the provided file path, creating and storing PartGeometry objects for each part in the _part_geometry_objects list. This class performs four major calculations which are Surface sampling, curve sampling, Winding number calculation and Derivative calculation. These are explained in detail in following subsections.

Surface sampling

The sample_surface function generates points from different types of surfaces (e.g, Plane, Cylinder, Sphere, Cone, Torus, and BSpline) and visualizes them using the meshplot library. We use different surface parameters for sampling, such as its location, axis, radius, and trim domain.

The function first validates the provided part and surface indices before extracting the corresponding surface patch. It then identifies the surface type and calculates the required parameters specific to that surface type. The function computes the u_values and v_values based on the trim domain and sample size, generating a set of sample points. It then calculates the 3D points for the surface type using the surface-specific equations.

The following is the pseudo code for sample_surface function:

Algorithm 2 sample_surface

```
1: procedure SAMPLE_SURFACE(orientation, surface_index, part_index, sample_size,
   plot_point_size, plot)
2:     Assert part_index and surface_index validity
3:     surface_patch ← corresponding surface from part
4:     surface_type ← surface_patch.type
5:     if surface_type is supported then
6:         Calculate parameters for surface_type
7:         Compute u_values, v_values, and sample_points
8:         Compute 3D points for surface_type
9:         if plot is not None then
10:            Add 3D points to plot
11:        else
12:            Create a new plot with 3D points
13:        end if
14:    else
15:        Raise ValueError("Surface type not supported")
16:    end if
17:    return surface_points, sample_points, plot, surface_patch
18: end procedure
```

Curve Sampling

The sample_curves3d function samples point from various types of 3D curves (e.g., Line, Circle, Ellipse, and BSpline) and visualize them using the mesh plot library. The input parameters of this function are namely: curve orientation, curve index, part index, and number of sample points. The curve gets processed depending on its type:

1. Line: The initial and final points are calculated, and a linearly spaced set of points is generated between them.

2. Circle: The circle points are calculated using radius, x-axis, y-axis, and location from the data. Then the points are generated by traversing this circle in a range specified by the curve interval.

3. Ellipse curves: Using the minor and major radii, ellipse centre, and X and Y-axes, the ellipse points are calculated. The points are then generated by traversing this ellipse in a range given by the curve interval.

4. BSpline curves: We use the geomdl library to create either a BSpline or NURBS curve, depending on the rational property of the curve. The curve points are then generated using the control points, knot vector, and weights (if applicable).

The function then reverses the point order if the provided orientation is False. Next, it creates edges between consecutive points in the list of curve points. After processing the input curve, the function returns the vertices (points), edge array, and plot object. The following visualization of a shape Cone 5.1 involves both curve and surface sampling.

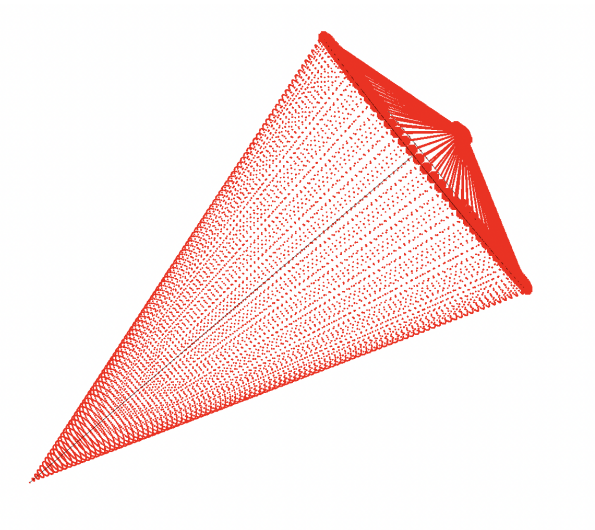

Figure 5.1 An example for Shape sampling

Winding Number Algorithm

The apply_winding_number_algorithm function calculates the winding numbers of a set of surface points with respect to a set of curve points on the surface. The input arguments of this function are 3D surface points, their corresponding 2D UV values, the 3D curve points, and optional period values for the U and V directions. The function returns the UV values of the curve points and the winding number list for the surface points.

First, we calculate the nearest points of the curve points in the surface points set using the calculate_nearest_points_of_setA_in_setB method. We use the cKDTree class from the SciPy library to retrieve the indices of these nearest points in the surface points set and extract their corresponding UV values. cKDTree constructs a k-dimensional tree for setB and performs a nearest neighbor search for each point in setA.

Next, we use compute_winding_number_vectorized method to calculate the winding number of the surface based on a set of associated points using vectorized implementation. The winding number is a topological invariant that quantifies how often a closed curve goes around a point on the surface. In this method, initially create a new set of associated

points that correspond to the periodicity of the surface. The function then iterates through the surface points and computes the winding numbers for each point. It calculates the dot product and determinant of vectors formed by subtracting the surface point from the adjacent curve points. Then, it calculates the winding number as the arctangent of the ratio of the determinant to the dot product. Finally, the winding number is divided by 2π to normalize it.

The following are the code snippets to compute Winding Number Algorithm:

```
def apply_winding_number_algorithm(self, surface_points, surface_uv_values,
    curve_points,period_u=None,period_v=None):


    nearest_points =
        self.calculate_nearest_points_of_setA_in_setB(curve_points,
        surface_points)
    curve_indexes = self.get_indexs_of_points_in_array(nearest_points,
        surface_points)
    curve_uv_values = surface_uv_values[curve_indexes.tolist()] # Getting uv
        values of the nearest points
    winding_number_list =
        self.compute_winding_number_vectorized(curve_uv_values,
        surface_uv_values,period_u,period_v)
    return curve_uv_values, winding_number_list
```

Vectorized version of the winding number algorithm:

```
def compute_winding_number_vectorized(self, associated_uv_valuesi,
    surface_uv_values,period_u=None,period_v=None):
    winding_number_list = np.zeros((surface_uv_values.shape[0], 1))
    associated_uv_values = associated_uv_valuesi
```

```
if period_u is None or period_v is None:
    if period_u is not None:
        u = associated_uv_valuesi[:, 0]
        v = associated_uv_valuesi[:, 1]

        associated_uv_values = np.array([np.sin(u / period_u * 2 * np.pi),
            np.cos(u / period_u * 2 * np.pi), v]).T

    if period_v is not None:
        u = associated_uv_valuesi[:, 0]
        v = associated_uv_valuesi[:, 1]

        associated_uv_values = np.array([u, np.sin(v / period_v * 2 *
            np.pi), np.cos(v / period_v * 2 * np.pi)]).T
elif period_u is not None and period_v is not None:
    u = associated_uv_valuesi[:, 0]
    v = associated_uv_valuesi[:, 1]

    associated_uv_values = np.array([np.cos(u / period_u * 2 * np.pi) *
        np.cos(v / period_v * 2 * np.pi),
                                     np.cos(u / period_u * 2 * np.pi) *
                                         np.sin(v / period_v * 2 * np.pi),
                                     np.sin(u / period_u * 2 * np.pi)]).T


for index, point in enumerate(surface_uv_values):
    position = point
```

```
        if period_u is not None:
            u = point[0]
            v = point[1]

            position = np.array([np.sin(u / period_u * 2 * np.pi), np.cos(u /
                period_u * 2 * np.pi), v])

        a = associated_uv_values[0:-2, :] - position
        b = associated_uv_values[1:-1, :] - position

        # Calaculate determinant of a and b
        if a.shape[1] == 2:
            det = a[:, 0] * b[:, 1] - a[:, 1] * b[:, 0]
        else:  # 3d
            det = np.linalg.norm(np.cross(a, b), axis=1)

        # Calculate dot product of a and b
        dot = a[:, 0] * b[:, 0] + a[:, 1] * b[:, 1]
        if a.shape[1] == 3:
            dot += a[:, 2] * b[:, 2]

        winding_number = np.arctan2(det, dot)
        winding_number = np.sum(winding_number)
        winding_number_list[index] = winding_number / (2 * np.pi)
    return winding_number_list
```

Derivatives and Normal Calculations

We calculate derivatives by processing surfaces in "part_geometry.surfaces," identifying types such as Plane, Cylinder, Sphere, Cone, Torus, or BSpline.

Algorithm 3 Calculate Derivatives and Normal Vectors

procedure CALCULATE_DERIVATIVES($self, order, part\_geometry, plot$)
    for each $surface$ in $part\_geometry$ **do**
        if $surface\_type$ is in supported types (Plane, Cylinder, Sphere, Cone, Torus, BSpline) **then**
            Compute partial derivatives of surface's parametric equation with respect to u and v
            Calculate normal vectors by taking cross product of partial derivatives
            Add normal vectors to the given *plot* object
        end if
    end for
    return updated *plot* object
end procedure

We compute the parametric equation and its derivatives and plot normal vectors if needed. Two helper functions assist in this process. The first calculates derivative and normal points, taking four inputs: equation variables, their values, derivative variables, and nth_order_equations. It ensures matching lengths, uses numpy and sympy for calculations, and returns appropriate values. The second function, "get_nth_order_equation," determines the nth-order derivative equation based on input parameters and handles exceptions. It calculates the derivative equation for one or two derivative variables accordingly.

The following are the code for the utility functions we created for derivative and normals calculations:

```
def get_nth_order_equation(self, order, equation, derivative_variables=[]):
    if order == 0:
        return equation
    else:
        if len(derivative_variables) == 0:
            raise Exception("No derivative variables")
        elif len(derivative_variables) == 1:
            derivative_equation = equation
            for i in range(order):
                derivative_equation = sympy.diff(derivative_equation,
                    derivative_variables[0])
            return [derivative_equation]
        elif len(derivative_variables) == 2:
            derivative_equation_1 = equation
            derivative_equation_2 = equation
            for i in range(order):
                derivative_equation_1 = sympy.diff(derivative_equation_1,
                    derivative_variables[0])
                derivative_equation_2 = sympy.diff(derivative_equation_2,
                    derivative_variables[1])
            return [derivative_equation_1, derivative_equation_2]
        else:
            raise Exception("Too many derivative variables")
```

```
def get_derivative_and_normal_points(self, equation_variables,
    equation_variable_values, derivative_variables,
                                    nth_order_equations):
    cross_product_values = []
    n_th_order_equation_values = []

    assert len(equation_variables) == len(equation_variable_values)

    # Calculating Points for derivatives
    for equation, variable in zip(nth_order_equations, derivative_variables):
        numpy_equation = sympy.lambdify(equation_variables, equation, 'numpy')
        numpy_equation_values =
            np.array(numpy_equation(*equation_variable_values)).T
        n_th_order_equation_values.append(numpy_equation_values)

    # Check if all elements are zero for n_th_order_equation_values
    if all([np.allclose(n_th_order_equation_values[i],
        np.zeros(n_th_order_equation_values[i].shape)) for i in
            range(len(n_th_order_equation_values))]):
        return [], []

    # Calculating the cross product of the derivatives
    if len(n_th_order_equation_values) > 1:
        for i in range(0, len(n_th_order_equation_values), 2):
            cross_product = np.cross(n_th_order_equation_values[i],
                n_th_order_equation_values[i + 1])
            cross_product_values.append(cross_product)
    else:
```

```
        cross_product_values.append(n_th_order_equation_values[0])

    return n_th_order_equation_values, np.asarray(cross_product_values)
```

The following image, Fig 5.2, is the snapshot for 1st order derivative and normal calculation of the Cylinder of revolution shape. The red points denote the normals to the surface.

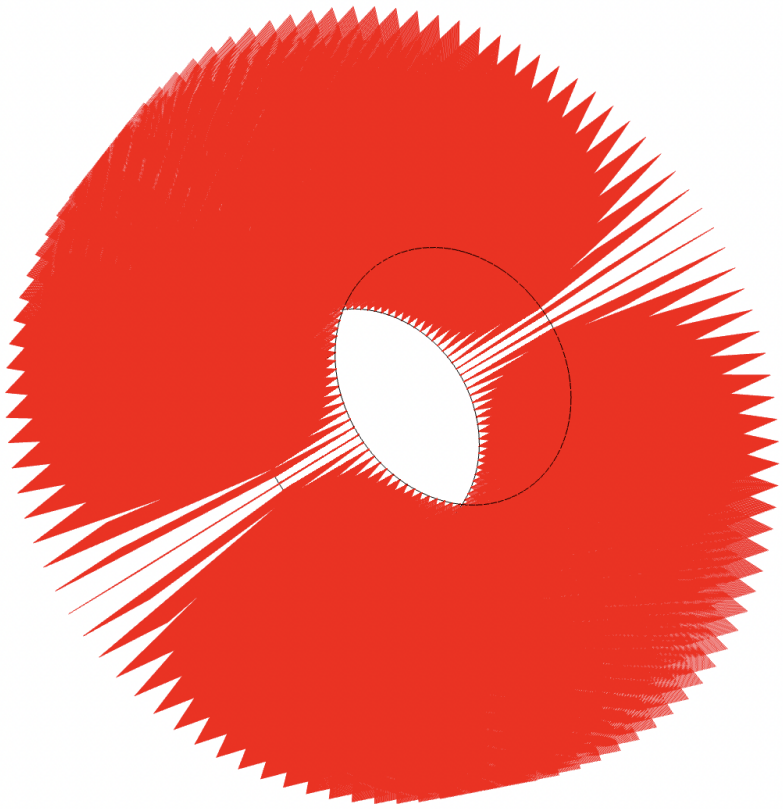

Figure 5.2 An example of sampled derivative and normal of a Cylinder of revolution

Chapter Six

Evaluation and Results

This chapter is an insight into the testing methods utilized for SmartCAD algorithm. There are four sections in this chapter. Section 6.1 mentions the processes used for the validation of algorithms accuracy and efficacy. Section 6.2 describes the techniques for optimizing the performance of the algorithm. Section 6.3 discusses the challenges during process implementation, and Section 6.4 shows snapshots of models developed using SmartCAD algorithm

6.1 Testing and Validation

Testing is important in any algorithm development to validate the algorithm's accuracy and efficiency. Various are tests are performed to ensure the conversion of successful sampling.

1. Visual inspection: Visual inspection involves plotting the sampled data alongside the original geometry (CAD model or mesh) to visually assess the distribution of sampled points and identify any apparent discrepancies or gaps in the data. This qualitative assessment can help identify potential issues with the sampling process or specific geometry regions that may require further attention.

2. Test Hausdorff distance [40]:

   It can be defined as the distance between a point p on a surface S and a surface S' is denoted as d(p, S'), where d represents the Hausdorff distance

$$d(p, S') = \min_{p' \in S'} \|p - p'\|^2$$

In this test, we measure the similarity between a sample point cloud from CAD models and its corresponding mesh data using the Hausdorff distance metric. The Hausdorff distance quantifies the degree of resemblance between two sets of points, with a smaller value indicating a closer match between the sampled data and the reference mesh. Through this test, we demonstrate the accuracy of the sampling process.

3. The Surface patch and loop adjacency Tests: This combined test validates the adjacency relationships and surface boundaries in a 3D model using Geometry and Topology Data. It involves performing a Surface Patch Test by determining the surface boundaries and sampling points on each surface to ensure they lie within the boundaries. Additionally, it conducts a Loop Adjacency Test by iterating through all edges and faces of a shape, identifying adjacent faces, and verifying their adjacency by checking the shared vertices. The correctness of the geometry data and their adjacency relationships can be confirmed by ensuring that all surfaces, edges, and adjacencies are consistent with the geometry model. This combined test is crucial for maintaining the accuracy and data integrity of the geometry model. Our findings showed that the threshold value for the average Hausdorff distance was 30, indicating a close resemblance between the sampled data and the reference mesh, and confirming the accuracy of the sampling process.

4. Distribution of sampled points: This analyzes the sampled points distribution on the geometry. For this test it requires that the points be evenly spread across the surface for specific geometry types like surfaces. An even distribution of points ensures that the sampled data accurately represent the geometry and minimizes potential errors in subsequent analyses or applications.

For the sample dataset, which includes a variety of shapes such as boxes, circles, cones, cylinders, and ellipses, we has successfully performed the necessary tests, including the Surface Patch and Loop Adjacency Tests as well as the Hausdorff distance evaluation. These tests have confirmed the accuracy and data integrity of the 3D models for the sample shapes.

In parallel, our team is working on another project, where we are in the process of converting a massive dataset of 1 million data points. The proposed tests, designed to ensure the accuracy and data integrity of the 3D models, include the Surface Patch and Loop Adjacency Tests, along with the Hausdorff distance evaluation. However, since we do not have the samples for these other models yet, we are unable to include the results of the tests conducted on them at this time. Once the data becomes available, our team will perform the necessary tests to validate the geometry and topology of those models as well.

The following image is an example image for the visual inspection and Distribution of sampled points of a hollow cylinder.

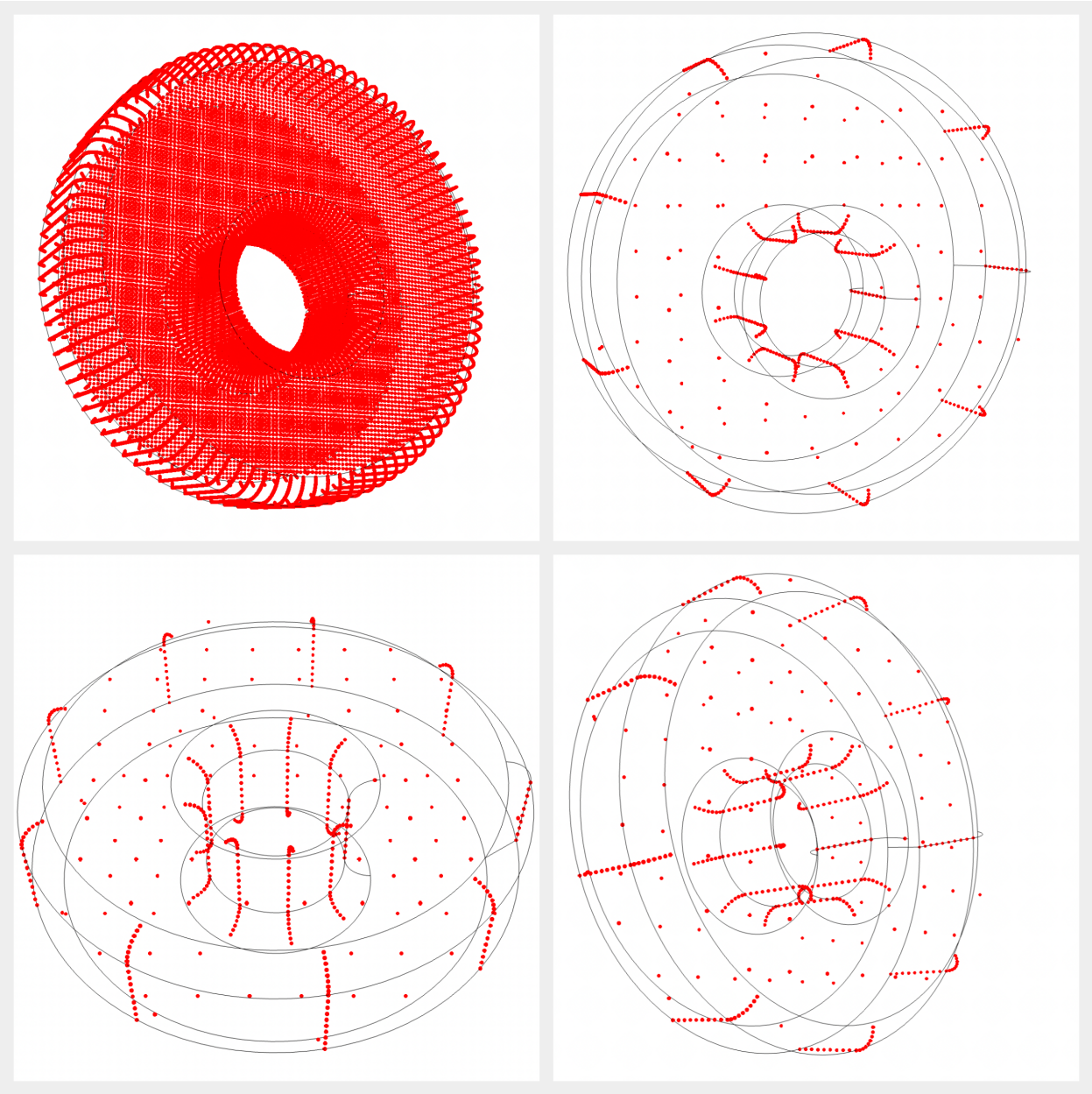

Figure 6.1 A Sample distribution for hollow cylinder from left to right at the top is densely sampled, and others are various angles of lightly sampled points

6.2 Optimization

We use several optimization processes to enhance the algorithm's performance and accuracy. The following lists some of the techniques we used :

1. Code Optimization:

    - Efficient data structures: Utilized NumPy arrays for faster computation of normal vectors and their cross products.
    - Vectorized operations: Replaced for-loops with NumPy vectorized operations when calculating the values of nth-order equations (derivatives) and cross-product values.
    - Parallel processing: Used multiprocessing or multithreading to parallelize the calculation of normal vectors for different surface types, speeding up the overall process.

2. Parameter Tuning:

    - Number of samples: Adjusted the number of samples (points) taken from each surface to balance the trade-off between computation time and accuracy. For example, increasing the number of samples resulted in more accurate normal vectors but increased computation time.
    - Tolerance value: Tuned the tolerance value for determining when to stop iterating when calculating the normal vectors, balancing computation time and precision.

3. Caching and memoization:

    - Intermediate results: Cached the computed partial derivatives of the surface's parametric equations with respect to u and v for each surface type, so they didn't need to be recalculated each time the function was called.

- Memoization: Utilized memoization in the get_nth_order_equation function to store the computed nth-order derivatives for each equation and set of variables, preventing the need to recompute them in future calls.

4. Heuristic techniques:

   - Surface sampling: Employed a heuristic approach to sample points on surfaces based on their curvature or complexity, allocating more points to areas with higher curvature or complexity to improve the accuracy of the normal vector calculation.

6.3 Challenges and Solutions

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

1. Handling diverse surface types and complex geometry: The algorithm supports various surface types, including planes, cylinders, spheres, and complex geometries such as BSplines. We implemented dedicated surface processing methods for each surface type and utilized polymorphism to ensure the correct method was called. We also developed specialized strategies to accurately calculate normal vectors for intricate geometries, such as BSplines, by studying their mathematical formulations, implementing appropriate calculations for their partial derivatives and normal vectors, and using additional libraries.

2. Balancing performance and accuracy: The balance between the algorithm's performance and the accuracy of the results was essential. We achieved this by fine-tuning parameters, such as the number of samples and the tolerance value, and employing heuristic techniques to optimize the algorithm's behaviour based on the surface properties and complexity.

3. Robustness and error handling: We ensured the algorithm was robust and capable of handling unexpected situations, such as invalid input data or inconsistencies in the CAD model, which is vital for practical applications. We incorporated comprehensive error handling, input validation, and defensive programming techniques throughout the implementation.

6.4 Sampling Results

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. Shape Cone

   The first example is a cone model, as shown in Figure 6.2. This figure illustrates the SmartCAD Algorithm's capability to accurately represent and process the cone's geometry using the YAML CAD data.

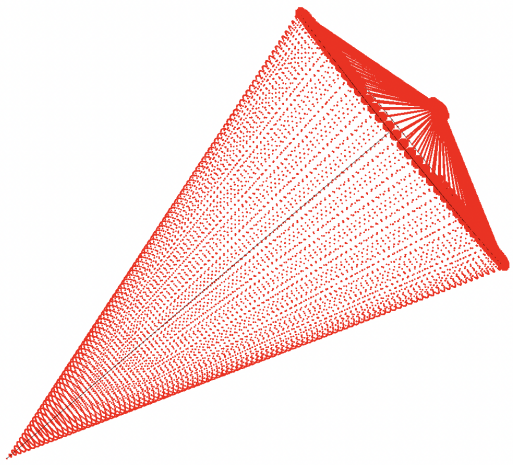

Figure 6.2 A Cone Model from YAML CAD data

2. Different models of cylinders with different sampling:

   We also tested the algorithm on various cylinder models with different sampling. Figures 6.3 to 6.6 demonstrate the algorithm's effectiveness in handling different cylinder shapes and complexities, including a cylinder wedge, a standard cylinder, and a cylinder with a hole and fillet.

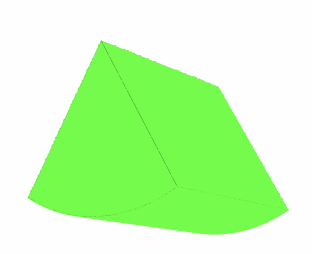

Figure 6.3 A Cylinder Wedge model from YAML CAD data

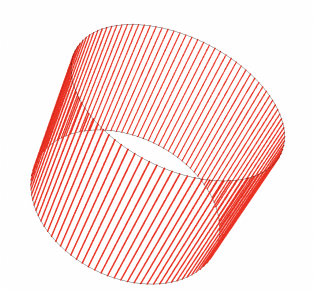

Figure 6.4 A Cylinder model from YAML CAD data

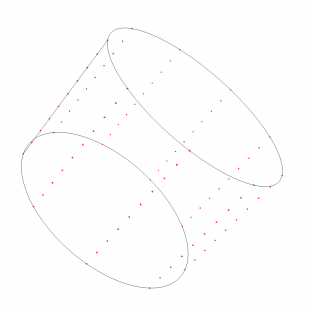

Figure 6.5 A Cylinder model from YAML CAD data

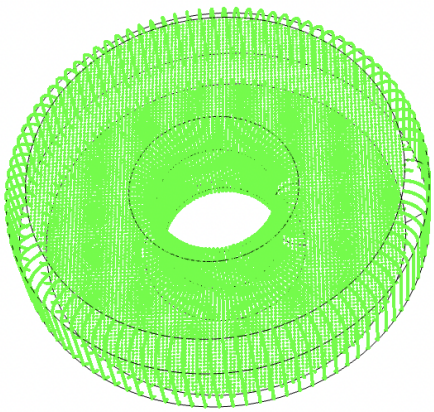

Figure 6.6 A Cylinder Hole Fillet Model from YAML CAD data

3. Complex shape: Bevel Gear

   In addition to basic shapes, the SmartCAD Algorithm is capable of processing complex geometries, such as a bevel gear model. Figures 6.7 to 6.9 showcase different views of the bevel gear model, demonstrating the algorithm's ability to accurately represent complex geometries using YAML CAD data.

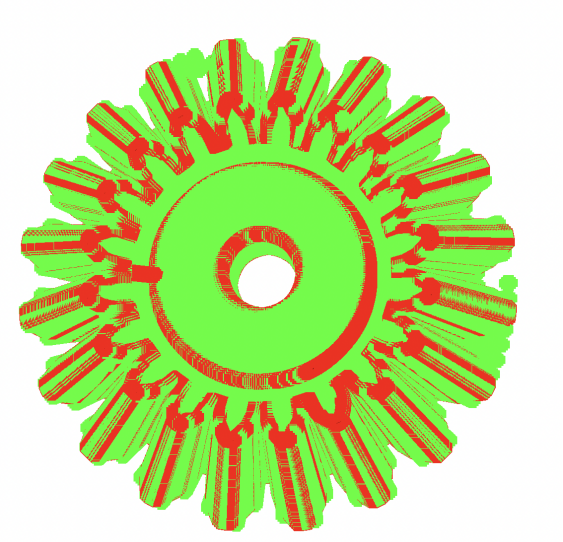

Figure 6.7 A Bevel Gear Model from YAML CAD data

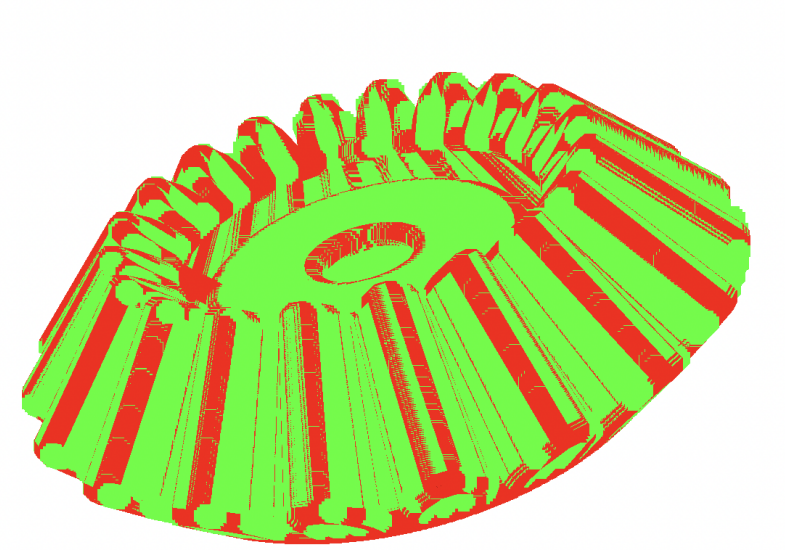

Figure 6.8 A Bevel Gear Model (top view) from YAML CAD data

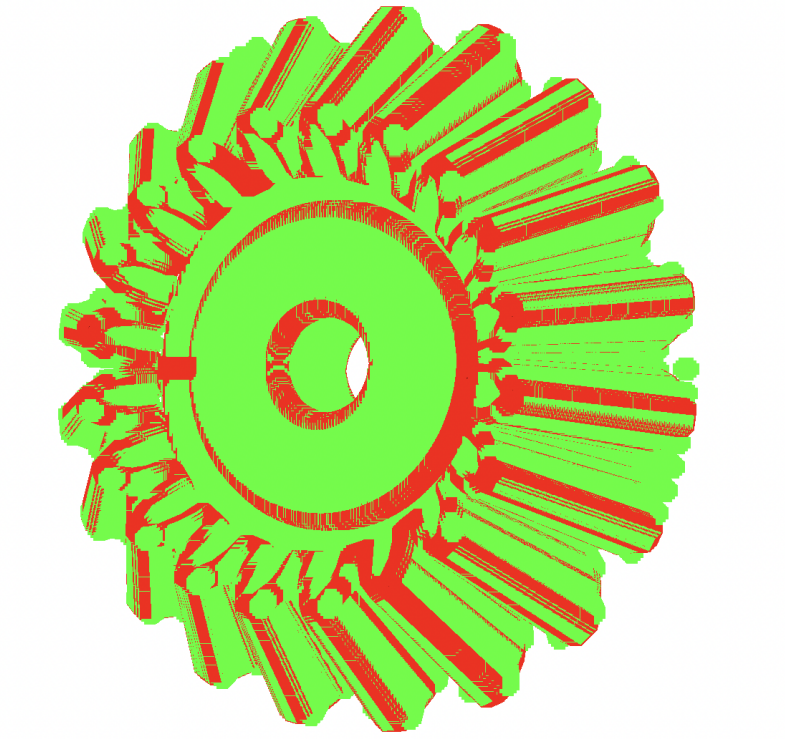

Figure 6.9 A Bevel Gear Model (front view) from YAML CAD data

4. Box

   Lastly, the SmartCAD Algorithm was tested on a box model, as shown in Figure 6.10. This example further emphasizes the algorithm's versatility and accuracy when processing and extracting information from various CAD geometries in the YAML format.

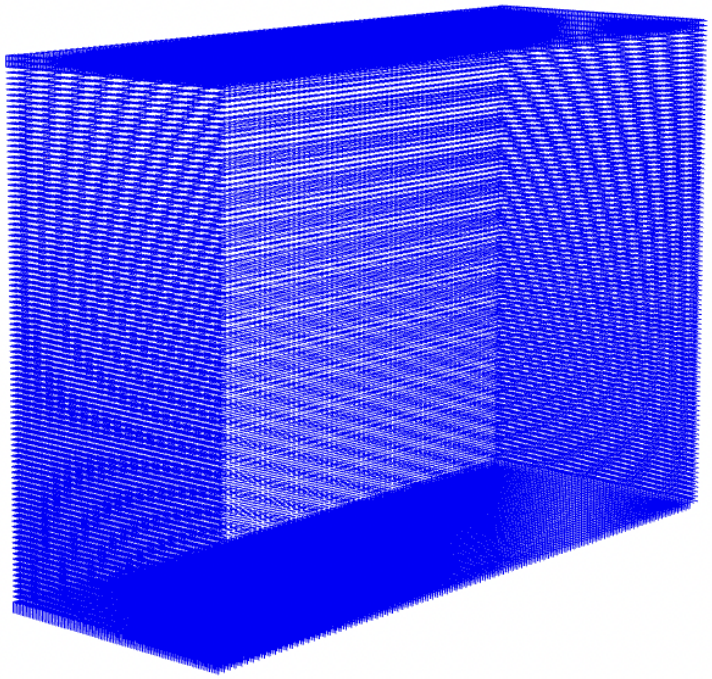

Figure 6.10 A Box Model from YAML CAD data

Chapter Seven

Conclusion and Future Work

7.1 Summary of Contributions and Findings

This project aims at addressing the various challenges related to CAD file conversion and the complexity of data organization in CAD systems. The developed algorithm facilitates efficient sampling and extraction of information from CAD files in the open-source YAML format, thereby providing a more accessible and versatile data structure. The algorithm demonstrates its effectiveness in handling the converted YAML models, paving the way for broader applications in the CAD domain. This project's results significantly contribute to improving CAD system efficiency and interoperability, ultimately benefiting various industries that rely on CAD for design and manufacturing processes.

7.2 Limitations and Future Work

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

1. Code Modularity: To ease the maintenance of the software and the scalability of the algorithm, there is a need to improvise code modularity. This shall enable the integration of the algorithm with other systems with ease and simplify the algorithm's process

of updating functionality.

2. Adaptive step size: Implementing an adaptive step size approach when calculating partial derivatives, adjusting the step size based on the surface's properties or curvature. This heuristic technique allowed for faster convergence without significantly compromising accuracy.

3. Implementing CI/CD for future versions: Implementation of Continuous Integration and Continuous Deployment pipeline, i.e. CI/CD shall ensure that the algorithm is up-to-date and reliable. CI/CD shall enable the rapid iteration and development of new features or improvements, thereby also ensuring that the project remains relevant and can be used by the CAD community.

4. Conversion of ABC dataset of 1 million CAD models to open-source YAML format: By converting the ABC dataset of 1 million CAD models to open-source YAM formats shall be a valuable resource to the data science community and thus set a new benchmark for evaluating new methods and algorithms.

5. Using the algorithm for sampling and data extraction from converted YAML models: Given the potential, this algorithm can be used as a primary tool to sample and extract information from the converted YAML models. Further research study shall help better the algorithm in terms of optimizing this algorithm with respect to its performance and broadening its applications to multiple CAD-related programs.

7.3 Conclusion and Final Remarks

In conclusion, this project successfully addressed the challenges associated with CAD file conversion and data organization. By developing a novel algorithm that efficiently samples and extracts information from CAD files in the open-source YAML format, the project

aims to make a significant contribution to the CAD domain. The outlined future work will continue to refine and expand upon the project's achievements, ensuring the developed algorithm remains a valuable resource for the CAD community and various industries that rely on CAD systems for design and manufacturing processes.

REFERENCES

[1] Ibrahim Zeid. *Mastering CAD/CAM*. 1st. McGraw-Hill Science/Engineering/Math, 2004. ISBN: 978-0072868456.

[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] Sharon J Kemmerer. "STEP: the grand experience". In: (1999).

[4] Aristides AG Requicha and Herbert B Voelcker. "Solid modeling: a historical summary and contemporary assessment". In: *IEEE computer graphics and applications* 2.02 (1982), pp. 9–24.

[5] Vadim Shapiro. "Solid Modeling". In: Jan. 2002, pp. 473–518. ISBN: 9780444511041. DOI: 10.1016/B978-044451104-1/50021-6.

[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*. 2019, pp. 9601–9611.

[7] Karl DD 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), pp. 1–24.

[8] Ian Stroud. *Boundary representation modelling techniques*. Springer Science & Business Media, 2006.

[9] Aristides G Requicha. "Representations for rigid solids: Theory, methods, and systems". In: *ACM Computing Surveys (CSUR)* 12.4 (1980), pp. 437–464.

[10] Thomas W Sederberg, David C Anderson, and Ronald N Goldman. "Implicit representation of parametric curves and surfaces". In: *Computer Vision, Graphics, and Image Processing* 28.1 (1984), pp. 72–84.

[11] Thibault Groueix et al. "AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation". eng. In: *arXiv.org* (2018). ISSN: 2331-8422.

[12] Thibault Groueix et al. "3D-CODED: 3D Correspondences by Deep Deformation". eng. In: *Computer Vision – ECCV 2018*. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 235–251. ISBN: 3030012158.

[13] Nicolas Donati, Abhishek Sharma, and Maks Ovsjanikov. "Deep Geometric Functional Maps: Robust Feature Learning for Shape Correspondence". eng. In: (2020).

[14] Lars Mescheder et al. "Occupancy Networks: Learning 3D Reconstruction in Function Space". eng. In: *arXiv.org* (2019). ISSN: 2331-8422.

[15] Marco Attene. "A lightweight approach to repairing digitized polygon meshes". In: *The Visual Computer* 26 (Nov. 2010), pp. 1393–1406. DOI: 10.1007/s00371-010-0416-3.

[16] Lang Zhou et al. "Point cloud denoising review: from classical to deep learning-based approaches". In: *Graphical Models* 121 (2022), p. 101140.

[17] The CGAL Project. *CGAL User and Reference Manual*. 5.5.2. CGAL Editorial Board, 2023. URL: https://doc.cgal.org/5.5.2/Manual/packages.html.

[18] Alec Jacobson et al. "libigl: A simple C++ geometry processing library". In: *Google Scholar* (2013).

[19] Paolo Cignoni et al. "MeshLab: an Open-Source Mesh Processing Tool". In: *Eurographics Italian Chapter Conference*. Ed. by Vittorio Scarano, Rosario De Chiara, and Ugo Erra. The Eurographics Association, 2008. ISBN: 978-3-905673-68-5. DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

[20] Michael M. Bronstein et al. "Geometric Deep Learning: Going beyond Euclidean data". eng. In: *IEEE signal processing magazine* 34.4 (2017), pp. 18–42. ISSN: 1053-5888.

[21] Harold Scott MacDonald Coxeter. *Introduction to Geometry*. 2nd ed. New York: John Wiley and Sons, Inc., Mar. 9, 1989. 496 pp. ISBN: 0-471-50458-0 978-0-471-50458-0. URL: http://plouffe.fr/simon/math/Wiley%20-%20Coxeter%20-%20Introduction%20To%20Geometry.pdf.

[22] *An Introduction to Topology*. Oct. 2015. URL: https://uwaterloo.ca/pure-mathematics/about-pure-math/what-is-pure-math/what-is-topology#:~:text=Topology%20is%20the%20study%20of%20properties%20of%20geometric%20figures%20that%20remain%20unchanged%20under%20continuous%20deformations..

[23] Joseph O'Rourke. *Computational geometry in C*. eng. Second edition. Cambridge: Cambridge University Press, 1998. ISBN: 0-521-64976-5.

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

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

[26] Franco P. Preparata. *Computational geometry : an introduction*. eng. Texts and monographs in computer science. New York: Springer-Verlag, 1985 - 1985. ISBN: 0387961313.

[27] Martti Mäntylä. *An introduction to solid modeling*. Computer Science Press, Inc., 1987.

[28] Mark. de Berg. *Computational Geometry Algorithms and Applications*. eng. 3rd ed. 2008. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. ISBN: 3-540-77974-4.

[29] Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. "Robust Inside-Outside Segmentation Using Generalized Winding Numbers". In: *ACM Trans. Graph.* 32.4 (July 2013). ISSN: 0730-0301. DOI: 10.1145/2461912.2461916. URL: https://doi.org/10.1145/2461912.2461916.

[30] 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.

[31] Les Piegl and Wayne Tiller. *The NURBS Book*. eng. Second Edition. Monographs in Visual Communication. Berlin, Heidelberg: Springer Berlin / Heidelberg, 1996. ISBN: 3540615458.

[32] David F. Rogers. *Mathematical elements for computer graphics*. eng. 2nd ed. New York: McGraw-Hill, 1990 - 1990. ISBN: 0070535299.

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

[34] 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.

[35] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

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

[37] 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.

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

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

[40] Nicolas Aspert, Diego Santa-Cruz, and Touradj Ebrahimi. "Mesh: Measuring errors between surfaces using the hausdorff distance". In: *Proceedings. IEEE international conference on multimedia and expo*. Vol. 1. IEEE. 2002, pp. 705–708.