Research Project

Conformal Lattice Structure Modelling in ClassCAD

Author: Yohanes Sugiarto
Politeknik ATMI Surakarta

Partner: Prof. Dr. sc. tecn. Norbert Frei
Dr. Rainer Weigel

1 Table of Contents

2 Summary

Developments in additive manufacturing (AM) are not without reasons. AM has several advantages, it reduces product development time and costs, reduces tools usage, saves material, and reduce human intervention in the manufacturing process. In AM, product design is transferred directly to the AM machine for production. AM also opens the door for lattice structure to become more popular. There are no barriers for applying lattice structure into a product as long as it is produced via AM. Lattice structure allows the creation of products that are light yet strong and sturdy.

Lattice structure becomes a necessary feature in modern computer-aided design (CAD) applications. ClassCAD as a framework that provides services for CAD systems needs to include the lattice structure as one of its feature among the whole wide range selection of features and services. This project explored the possibility to realize that.

There are four types of lattice structure, i.e. conformal periodic, conformal non-periodic, non-conformal periodic, and non-conformal non-periodic. Periodic lattice structures use array of unit cells to construct the structures. In a non-conformal one, the unit cells maintain their form so trimming process is necessary to get the expected volume. In a conformal one, the unit cells are deformed more and more towards the volume surface, so that in the end the unit cells conform to the surface. Non-periodic lattice structures have no arrangement of unit cells, struts are joined at random points inside the lattice volume. In lattice structures where randomness and conformity become characteristic, meshes can be used to generate lattices.

Non-conformal periodic lattice structure can be constructed using the existing features of ClassCAD. But efficiency is still a big question. Other lattice structures that use meshes can be explored using PythonOCC and Meshio. The mesh can be generated using Gmsh application.

3 Introduction

3.1 Project Description

In the manufacturing industry, several conventional methods are common, namely subtractive manufacturing and formative manufacturing. Subtractive manufacturing is used to create products using material removal processes. A product is shaped by removing material through cutting, drilling, grinding, and eroding. Formative manufacturing is used to make products using forming processes, there is almost no material is added or removed in these processes. A product is shaped by stresses, like compression, shear, or deformation. Widely known forming process is injection molding which creates plastic products through injection of melted plastic material into the cavity of the mold.

Currently, another manufacturing method is on the rise and is trending, it is additive manufacturing (AM). As the name suggests, AM creates products by adding material. Material is stacked layer by layer in thin 2D cross sections until the final 3D product is completed. The technology that is very popular and has penetrated home industries is 3D printing. Actually, welding is also an additive method, because it fills small gaps with material, but nowadays there are 3D metal printers that use the principle of welding. The rapid use of AM is because this method has several advantages, including reduction of product development time and costs, saving of material, reduction of additional tools, such as cutting tools and molds, and reduction of human intervention in the process, because the design can be directly transferred to the AM machine for building. The most notable advantage is that AM allows the manufacture of products with complex geometries which are not possible with the two conventional methods previously mentioned. This allows the design processes to focus more on functional requirements.

At the same time, a design feature that is also currently gaining popularity is lattice structure. Lattice structure can be easily produced using additive methods. Lattice structure is usually found to fill in voids in 3d printing processes if there is a functional part on them. Today, the lattice structure is widely used to reduce the material or the weight and still maintaining the structural strength. Many computer-aided design (CAD) software facilitate the design of the lattice structure. Although the lattice structure looks complicated, CAD makes it possible to generate it automatically using certain parameters. ClassCAD, which is an object-oriented interpreter programming language for 3D modelling, needs to integrate the lattice structure feature.

As part of this project, the idea to incorporate the lattice structure creation in ClassCAD needs to be pursued further, all possibilities need to be studied, and a prototype need to be developed.

3.2 Goals

The main aim of this project is to explore and examine the possibility of modelling the lattice structure in ClassCAD and in ClassCAD-based applications.

The following tasks give an overview of the steps, studies, and areas to be explored in order to satisfy the main goals:

- Incorporate the study of related works and concepts that have been developed.
- Hands-on experience using existing systems that have integrated the lattice structure generation.
    - Steps and Procedures.

- Concepts and Structures.
- Feedbacks.

- Study the possibility to generate the lattice structure in ClassCAD.
  - Buerli-Modeler will be used to examine the implementation.
- Implementation of concepts, algorithms, and techniques in prototypes or functions.
  - Periodical and Non-Periodical Lattice Structure.
  - Conformal and Non-Conformal Lattice Structure.

3.3 Motivation

The inspiration for this project came from the idea of the president of AWV-Informatik AG, Mr. Rainer Weigel, to support lattice modelling in ClassCAD-based applications. The lattice structure is increasingly used in the design processes and integrated in many CAD applications. It can also be further used to analyse the functionalities and strengths..

4 Literature Review

4.1 Lattice Structure

The lattice structure has become very popular with the development of additive manufacturing. The lattice structure is applied in many fields, such as automotive, aerospace, mechanical engineering, biological engineering, medical, construction, etc. Where there is a need for high strength less weight components, the lattice structure becomes an answer.

The idea behind the lattice structure came from the crystal structure in nature[1]. The lattice structure is composed of groups of components in a repeating pattern. The group is called unit cell. In a crystal structure, the unit cell maintains its arrangement, making it homogenous. The unit cell contains some particles that fill the space of the unit cell. Most unit cells are hexahedral (6 faces). Figure 4-1 shows how particles can be arranged in a cubic unit cells.

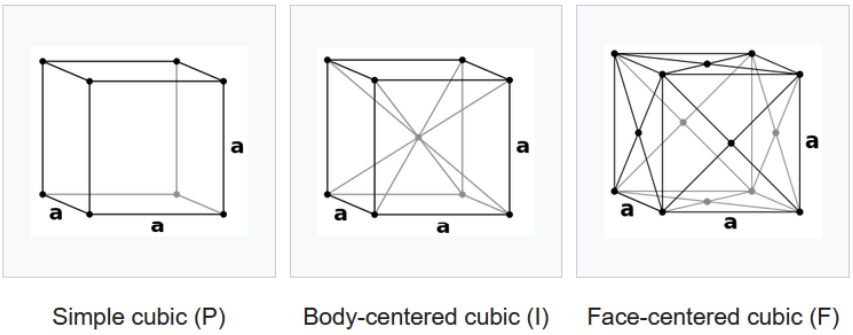

Figure 4-1. Particles Arrangement

In additive manufacturing, lattice structure is a type of architectured material formed also by an arrangement of unit cells. The unit cell consists of edges and nodes. The edges are the struts and the connections between those struts are the nodes.

4.2 Paper 1: Review on Characterization and Impact of The Lattice Structure in Additive Manufacturing

This paper highlights the importance of lattice structures in AM and how the AM paves the way in designing and modelling lattice structures. A lattice structure is an architecture formed by an array of spatial arrangement of unit cells with edges and faces. The lattice cellular materials are formed by the set of beam elements or supporting struts connected to nodes. There are two main topologies of lattice structure, namely stochastic and periodic. The shape and arrangement of unit cells in stochastic lattice structure are random, they have unsystematic probability distribution. Whereas the arrangement of unit cells in periodic lattice structure is fixed along axis.

Several CAD softwares that support generation of lattice structure mentioned in this paper are Solidworks, Meshmixer, and a Matlab lattice generator.

The lattice structure can be three times stronger than the ordinary structure if its volumetric density set correctly. The optimization of the lattice structure depends on homogenization and ground structure methods. Three types of stresses are possible to be loaded onto lattice structures which are bending stress, shear stress, and tension.

[1] https://en.wikipedia.org/wiki/Crystal_structure

Important conclusions from this paper:

- Additive manufacturing is the only well-suited method for printing lattice structured materials due to its complexity.
- It is found that lattices have created to form complex geometries with certain properties like less weight, high stiffness ,low relative density, larger elasticity, and strength compared to other solid structures.
- Lattice cellular materials can have high wear resistance and also reduces the manufacturing cost and time in industries.

4.3 Paper 2: Creation of Lattice Structures for Additive Manufacturing in CAD Environment

This paper presents a method for generating lattice structures and shows the implementation in Visual Basic using the API from SolidWorks. The main focuses on developing an application to generate lattice structure are:

- Creating a library of various types of basic unit cells.
- Generating lattice structure automatically.
- Changing the parameters of unit cells and lattice structures easily.

The method to create lattice structures consists following steps:

1. Create a library for unit cell selection.
2. Create a layer of unit cells.
3. Create a volume of lattice structure.
4. Trim the volume according to design space of the product.

Figure 4-2 shows the created library for unit cell selection.

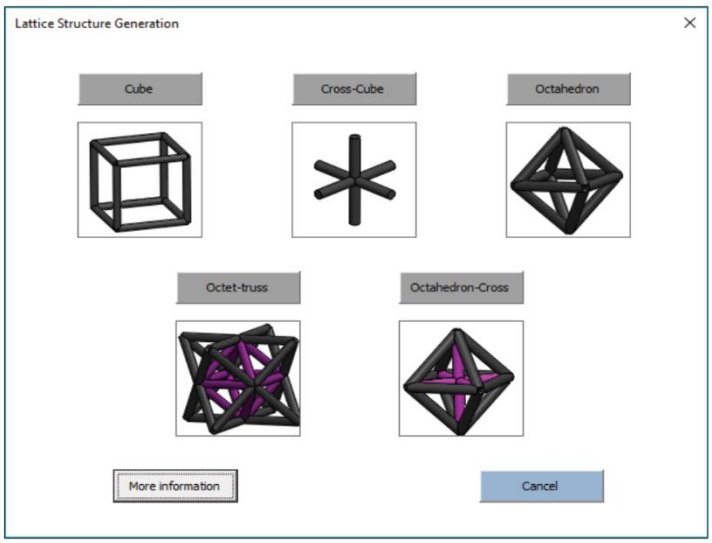

Figure 4-2. Unit Cells

Figure 4-3 shows the creation of volume of lattice structure, the trimming of lattice structure to fit the design space, and finally the assembly of the lattice structure in a product.

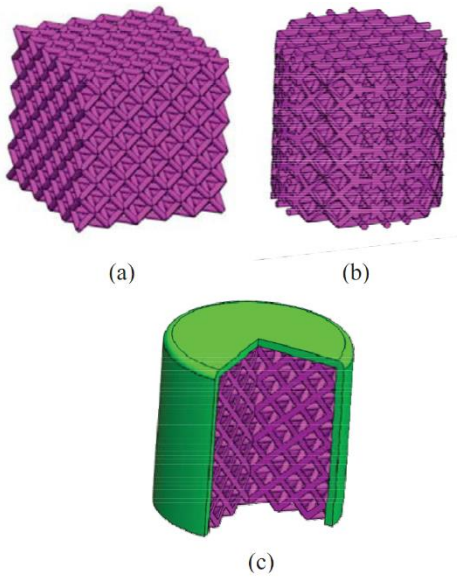

Figure 4-3. Lattice Volume (a), Trimming (b), and Assembly (c)

Important conclusions from this paper:

- Method describe in this paper is to generate homogenous periodic lattice structures. It is important to notice that 2D arrangement (single layer) of unit cells comes before 3D arrangement (volume) of the whole lattice structure.
- Basic types of unit cell in cubic form which can fill in space easily.
- Trimming makes use of cutting operation in SolidWorks.
- The problems for the future are the processing time and the memory consumption.

4.4 Paper 3: A Method to Generate Lattice Structure for Additive Manufacturing

This paper is basically similar to the second one but with some additional details and advancements. The generations of lattice structure are divided into two part, namely for a periodic lattice structure and a non-periodic lattice structure.

The generation of periodic lattice structure describes the creation of unit cell in more detail. The connection between struts needs to be filled with a sphere to avoid an empty space. The radius of the sphere should be at least equal to the radius of the strut. Figures 4-4 shows how the struts, the connection, and the sphere are combined to create joint.

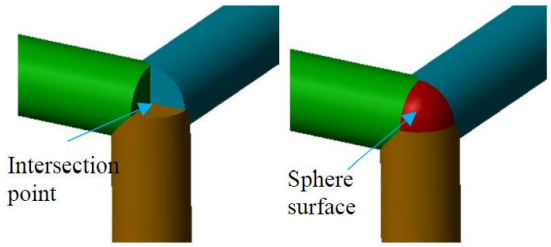

Figure 4-4. Adding sphere

Several types of unit cells are also added which are shown in Figure 4-5.

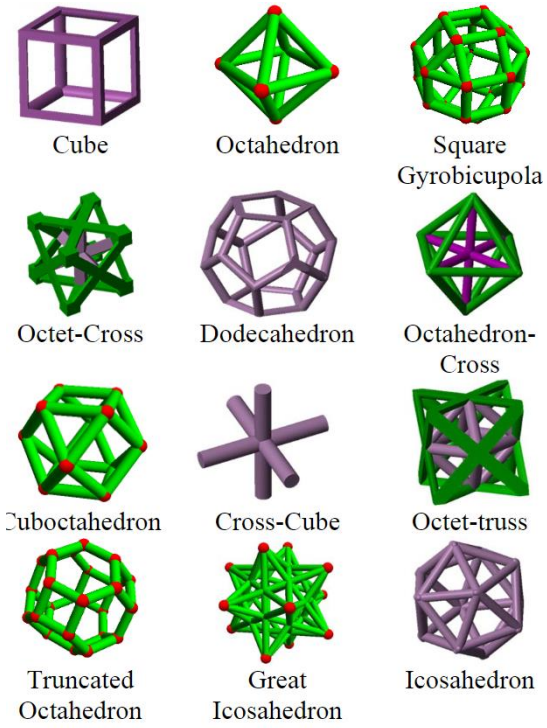

Figure 4-5. Unit Cells

The generation of non-periodic lattice structure is proposed as shown in Figure 4-6.

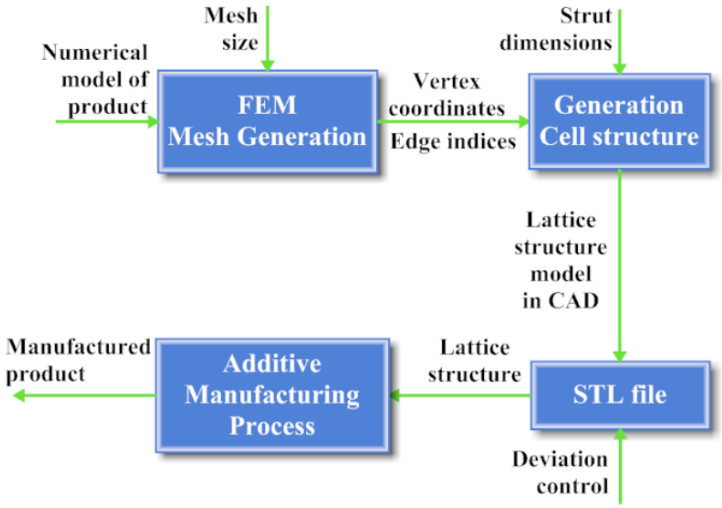

Figure 4-6. Non-periodic lattice generation

In the non-periodic lattice structure generation method, the mesh data containing vertex coordinates and edge indices is used to create the lattice structure. Each strut of the lattice structure is actually an edge connecting points of each meshing element. As in the periodic one, spheres must be added into the connections between struts. The paper shows several cases on PTC Creo Parametric software and the implementation of the generation of non-periodic lattice structure on Mathematica. Figure 4-7 shows how the mesh can be used to create lattice structures.

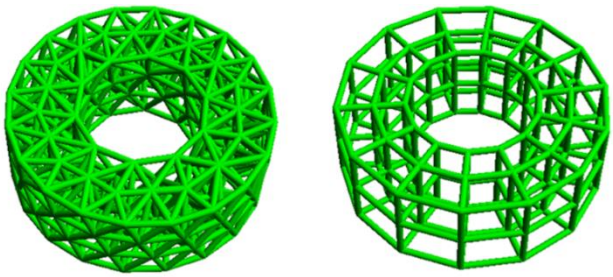

Figure 4-7. Conformal Lattice Structures

Important conclusions from this paper:

- Ensure that joints between struts are not empty.
- If the unit cell is broken down into connections of struts, periodic and non-periodic lattice structure could be generated in the similar fashion as long as the unit cell cube points (periodic) or mesh vertices (non-periodic) are available.

4.5 Paper 4: Conformal Lattice Structure Design and Fabrication

This paper provides methods for generating conformal lattice structure (CLS). Conformal lattice structure constructs the cellular materials or unit cells in such a way that they conform to the shapes of part surfaces. Figure 4-8 shows the difference between uniform and conformal lattices.

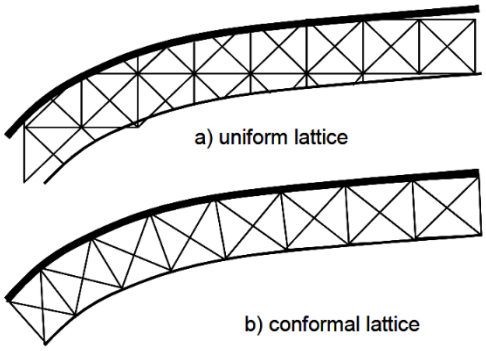

Figure 4-8. Uniform and conformal lattices

Looking back at the previous papers, the equivalent of a uniform lattice is a periodic lattice, and conformal lattice could be also a kind of non-periodic lattice with some degree of regularity, similar in term of Figure 4-8.b.

The first method presented in this paper is the construction of CLS. It basically uses the same concept as in Paper 3 for non-periodic lattice, namely using the mesh. However, it uses the conformal hexahedral mesh. A hexahedral mesh element has 6 faces and 8 vertices. Each unit cell or cell primitive will then be mapped to each mesh element.

The second method described in this paper is augmented size matching and scaling (SMS). This method use heuristic approach to optimize further the lattice structure based on loading conditions.

Important conclusions from this paper:

- Conformal lattice structure that has unit cell can be created using conformal hexahedral mesh.

4.6 Study on nTopology Application

nTopology is an application that focusing in the generative design. It has its own implicit modeling engine.

"Implicit modeling is a unique and light way of representing 3D objects using a single mathematical function to describe a solid body, both its external shape and its interior characteristics."[2]

It has some incredible capabilities inculding topology optimization, lattices, design automation, simulation, and field-driven design. This study focusing only on the lattices.

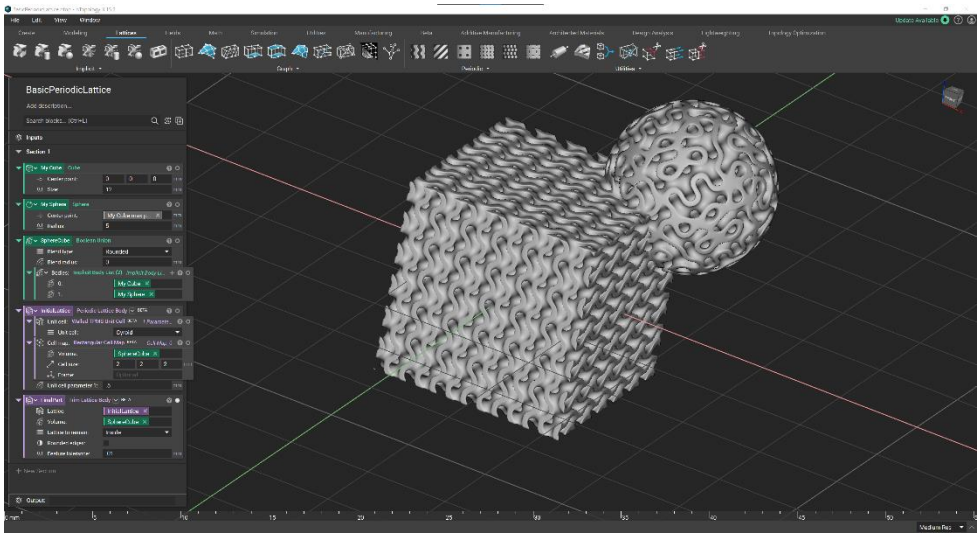

Figure 4-9. nTopology Application

nTopology treats features and operations as blocks which can be referenced as variables for multiple uses. Figure 4-9 shows a periodic lattice structure in a volume which is a combination between a cube and a sphere. The basic steps for creating such a lattice volume are as follows:

1. Create a primitive which can be considered as a solid.
2. Create a variable for the primitive.
3. Step 1-2 can be repeated as many times as needed. Figure 4-9 create two primitives, a cube and a sphere.
4. Create any operation between primitives if necessary.
5. Create variable for each operation, so it can be used in the next processes.
6. Create a periodic lattice body. Create variable for it.
    a. Select the type of the unit cell.
    b. Select the volume for the arrangement to be applied, bounding box will be used.
7. Create a trim lattice body for the finalization.
    a. Select the lattice body in step 6 as the lattice.
    b. Select the solid as the volume to trim the lattice.

[2] nTopology. "Generative Design with Complete Control".

The block structure for this example is shown below. The blocks are self explanatory, they reflect the needed steps.

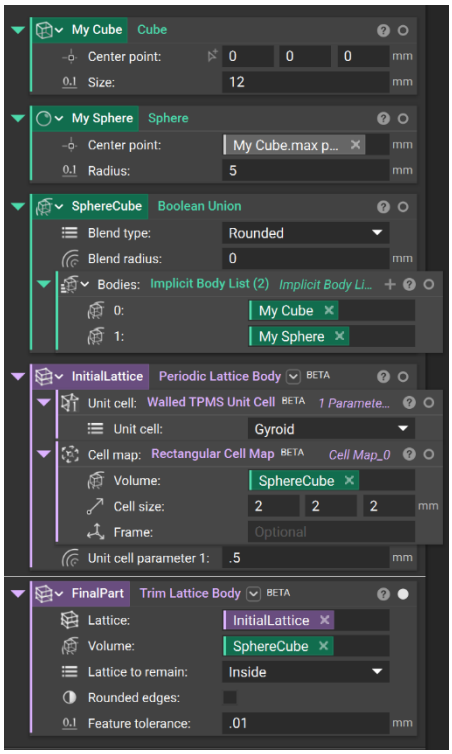

Figure 4-10. nTopology blocks

To create non-periodic lattice structure, the steps are similar. It is a matter of correct parameterization for each block. Following figures summarize the creation of non-periodic lattice structure using mesh.

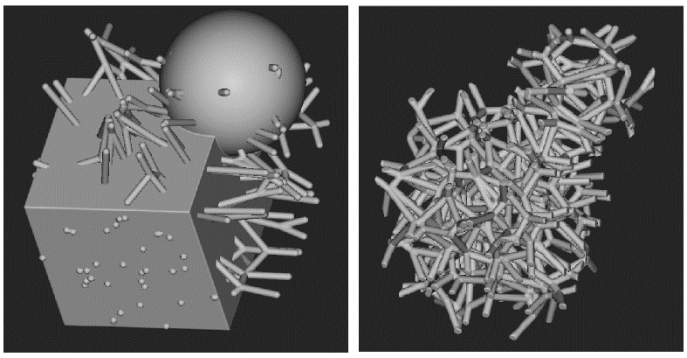

Figure 4-11. Non-periodic lattice structure

Figure 4-11 shows the position of non-periodic lattice structure in accordance with the volume. In this lattice, 500 beams or struts are created.

To create conformal non-periodic lattice structure, additional mesh creation step is needed. Following figures summarize the creation of conformal non-periodic lattice structure using mesh.

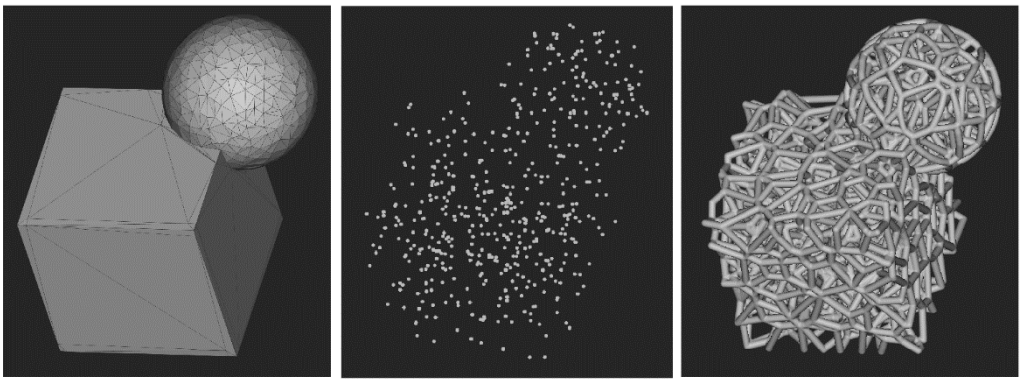

Figure 4-12. Conformal non-periodic lattice structure

Figure 4-12 on the left shows the created mesh, the created random points is in the middle, an on the right shows the created conformal non-periodic lattice structure. The beams on the surface follow the surface and there is no trimmed beams.

4.7 ClassCAD

ClassCAD is a framework that provides services to build CAD systems. ClassCAD uses object-oriented manner to manage its classes and objects. The object modelling and parameterization of CAD models are the strengths of ClassCAD. CAD systems based on ClassCAD can be built both as plug-in for other CAD systems as well as standalone CAD applications.

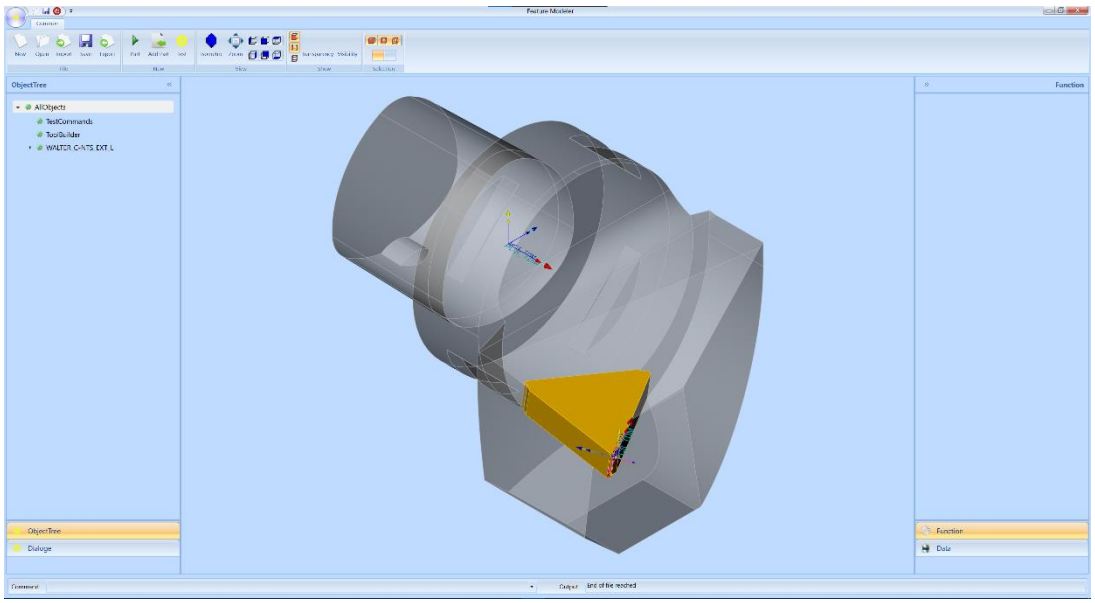

Figure 4-13. ClassCAD Feature Modeler

Figure 4-13 shows a standalone CAD application based on ClassCAD built on .NET Framework. ClassCAD has its own properietary programming language. ClassCAD programming language is an object-oriented, interpreted language.

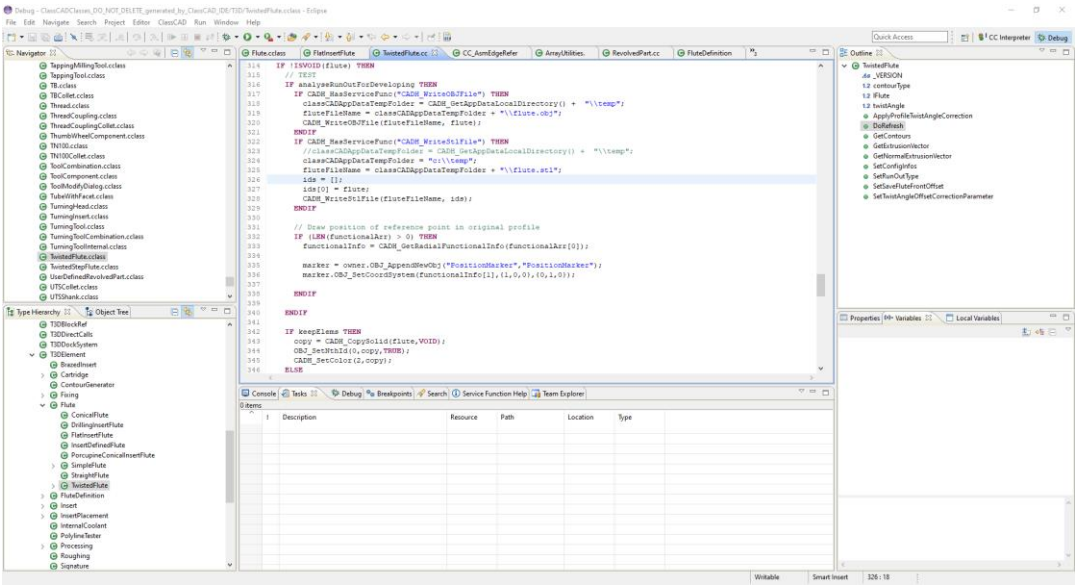

Figure 4-14. ClassCAD programming

Figure 4-14 shows a glimpse of ClassCAD programming language in Eclipse IDE. ClassCAD programming language facilitates the development and customization of CAD functionalities. Building CAD systems has never been easier.

4.7.1 Buerli

Buerli is a framework that supports CAD system based on ClassCAD for the web. It has a lot of features that allows the development of interactive browser based CAD applications.

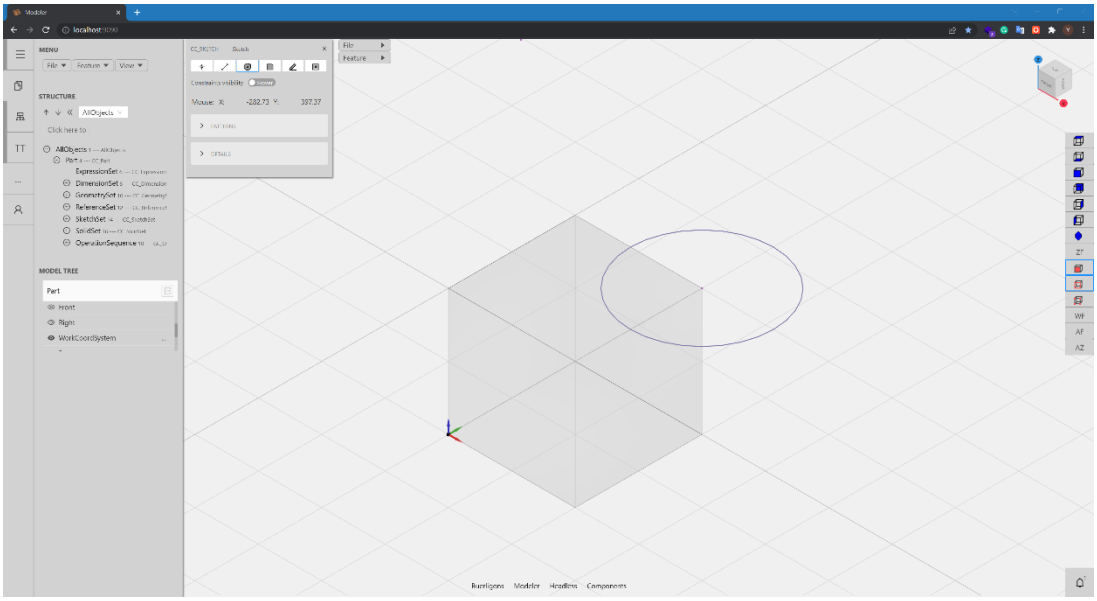

Figure 4-15. Buerli-Modeler

Figure 4-15 shows a Buerli-Modeler in action, creating part and sketch interactively. In this project the Buerli-Modeler is used. However, the underlying technology is ClassCAD and the logic behind it is also written using ClassCAD programming language.

4.8 Open CASCADE

Open CASCADE Technology (OCCT)[3] is a 3D modeling kernel which is freely available in open source. OCCT is an object-oriented C++ class library which provides service for:

- Basic data structures (geometric modeling, visualization, interactive selection and application specific services).
- Modeling algorithms.
- Working with mesh (faceted) data.
- Data interoperability with neutral formats (IGES, STEP).

Figure 4-16 shows the modular structure of OCCT. There are seven modules that can be used to develop CAD applications.

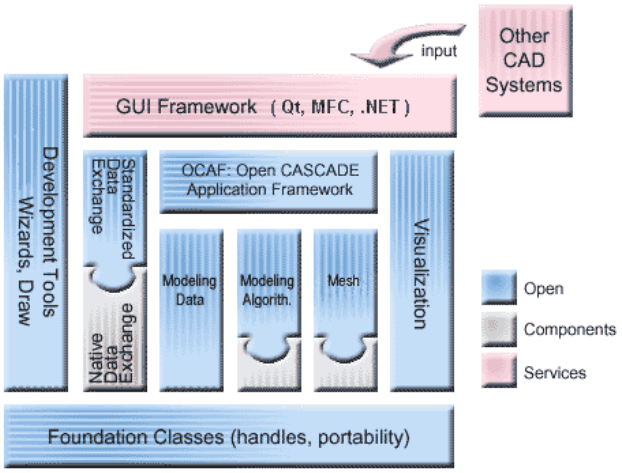

Figure 4-16. OCCT Modules

- Foundation Classes module underlies all other OCCT classes.
- Modeling Data module supplies data structures to represent 2D and 3D geometric primitives and their compositions into CAD models.
- Modeling Algorithms module contains a vast range of geometrical and topological algorithms.
- Mesh toolkit from "Modeling Algorithms" module implements tessellated representations of objects.
- Visualization module provides complex mechanisms for graphical data representation.
- Data Exchange module inter-operates with popular data formats and relies on Shape Healing to improve compatibility between CAD software of different vendors.
- Application Framework module offers ready-to-use solutions for handling application-specific data (user attributes) and commonly used functionality (save/restore, undo/redo, copy/paste, tracking CAD modifications, etc).

[3] https://dev.opencascade.org/doc/overview/html/index.html

4.8.1 PythonOCC

PythonOCC[4] is an open source Python-wrapper for Open CASCADE. It provides full access from Python to almost all of the OCCT C++ classes. All classes and methods share the same names and also the same signatures. With this wrapper, CAD and geometry scripts can be written in Python, which is an interpreted object-oriented scripting language. PythonOCC is used to speed up the prototyping and concept proofing processes.

PythonOCC provides many examples which can be found in its Github repository[5]. PythonOCC can also be used on Jupyter Notebook, making it quite easy to test concepts and build prototypes.

[4] https://github.com/tpaviot/pythonocc

[5] https://github.com/tpaviot/pythonocc-demos

5 Methodology

5.1 Overview

Lattice structure is a type of architectured material composed of repetition of solid and empty space. Lattice structure is a kind of porous material, this is one generic characteristic of the lattice structure. Furthermore, lattice structure has some specific characteristics, depending on how it is constructed. The following table describes the specific characteristics based on the categorization of the lattice structure:

Table 1. Category of Lattice Structure

Lattice Structure	Periodic	Non-Periodic
Conformal	• Use unit cell • Use mesh down to the volume and surface • No trimming • Unit cell and mesh structures must match, because each one of them will be mapped onto the other	• No unit cell • Use mesh down to the volume and surfaces • No trimming
Non-Conformal	• Use unit cell • No mesh • Trimming necessary • Unit cell multiplied and arranged in 3D direction to fill the bounding box of the volume	• No unit cell • Use mesh to the bounding box of the volume, surfaces are not considered • Trimming necessary

What is important to notice from Table 1 is that some lattice structure constructions require meshes and one does not. Considering the capability of ClassCAD in current state, the non-conformal periodic lattice structure is implemented directly in ClassCAD at the first stage. Further stages which require mesh generation are implemented using PythonOCC.

5.2 Technologies

Several technologies are used in this project. The selection is based on the accessibility, the availability, and the functionality, especially for rapid prototyping of functions and applications. However, in accordance with the main goals of the project, the ClassCAD implementations should get the priority.

5.2.1 ClassCAD – Buerli

Buerli is a framework that expanses the ClassCAD capabilities to be used as web-based 3D modeling. The main components are all in ClassCAD. The third-party libraries are also wrapped and provided as ClassCAD services.

Buerli-Modeler is used as the tool to test the visualization and the structure of implemented ClassCAD classes and methods.

Development Environment

In order to develop using ClassCAD, the whole ClassCAD base solution is downloaded and built using Microsoft Visual Studio 2017. In this project, this is done only once to localize the development changes in ClassCAD. It should prevent unnecessary conflicts or instability, especially for the master branch or head because ClassCAD is always in a state of rapid development and new features are added continuously. The build process generates the core packages for further use in Buerli.

Buerli-Modeler needs to be built by involving several Buerli packages on which it depends on. The packages used for this purpose are:

- buerli
- buerli-react-cad
- buerli-tooling
- buerli-modeler
- buerli.io
- buerli-backend
- buerli-developer
- buerligons
- icons

Microsoft Visual Studio Code is used to build the Buerli-Modeler, however the pre-requisites must be installed first, namely:

- Node.js
- NPM (Node Package Manager)
- Yarn Package Manager

The configurationData and output entries in server.js for the Buerli-Modeler should point to some items generated from ClassCAD build.

```
configurationData: 'BuerliDemo.ini',
output: 'D:/workspaces/mybuerli/Output/VS2017_64/Release',
```

Furthermore, BuerliDemo.ini in the generated release folder from ClassCAD build should load the necessary packages. BuerliDemo.ini used in this project is shown below.

```
1. [HKEY_LOCAL_MACHINE\SOFTWARE\AWV\ClassCAD\Common]
2. // ShowClasses enables the developer mode
3. "ShowClasses"="1"
4. "dataPath"="(APP_DIR)data"
5. "verboseFileName"="(APP_DIR)LogFiles\BuerliDemoLog.txt"
6. "verboseLevel"="16583"
7. "basedll"="ExpAP.dll"
8. "basedlld"="ExpAPd.dll"
9. "tessellateCurves"="0"
10. "CC_ProductRefCreation"="3"
11. //Set this to "1" to enable protocol and logic changes for awv3-reboot / buerli
12. "clientRebootEnabled"="1"
13. "classpath"="(APP_DIR)..\..\..\cclasses"
14. //Testing
```

```
"testInputPath"="(APP_DIR)..\..\..\Data\Test\Input"
"testOutputPath"="(APP_DIR)..\..\..\Data\Test\Output"

[HKEY_LOCAL_MACHINE\SOFTWARE\AWV\ClassCAD\SMLib]
"loadClassesFromDirectory"="1"
"classfile"="(APP_DIR)data\BuerliDemo.cf1"
"packagesToLoad"="System;System.GUI;Common;BaseModeling;BaseModeling.CCBaseModeling.*;FeatureModeler.*;T3D.*;CADModeler;GTC.*;ISO_Tool.*;CAD;Sheet;ISOChecker"
//"servicedll"="SMLibService.dll;SMLibExpressService.dll;GenericService.dll;LGSService.dll;LGS3DService.dll;BrepSimplificationService.dll;"
"servicedll"="SMLibService.dll;SMLibExpressService.dll;GenericService.dll;LGSService.dll;LGS3DService.dll;BrepSimplificationService.dll;IDEConnectionService.dll;RapidXMLService.dll"
"servicedlld"="SMLibServiced.dll;SMLibExpressServiced.dll;GenericServiced.dll;LGSServiced.dll;LGS3DServiced.dll;BrepSimplificationServiced.dll;IDEConnectionServiced.dll;RapidXMLServiced.dll"
"applicationClass"="BuerliDemoApp"
```

To build the Buerli-Modeler, point the terminal in Visual Studio Code to the buerli-modeler working directory and run the command:

```
yarn i:a
```

Still in the same working directory, to start the server run the command:

```
yarn classcad
```

To connect to the ClassCAD development, open new terminal in the same working directory and run the command:

```
yarn dev
```

The ClassCAD development is done using Eclipse IDE. There is a plugin for ClassCAD development in Eclipse and currently it is running stable in Eclipse Kepler.

Implementation

Currently, the implementation of ClassCAD is so advanced that it incorporates the structure of part and assembly. The construction of lattice structure will usually be applied on the part level. This narrows the focus of work on the structure of part.

All operations on the part handled by CC_Operation. CAD features are also operations. Therefore, both the lattice structure as well as the unit cell must be derived from CC_Operation. In this stage, a non-conformal periodic lattice structure is implementes. A unit cell takes a shape of a cube, so it is better to take it a little further down as a descendant of CC_BasicShape.

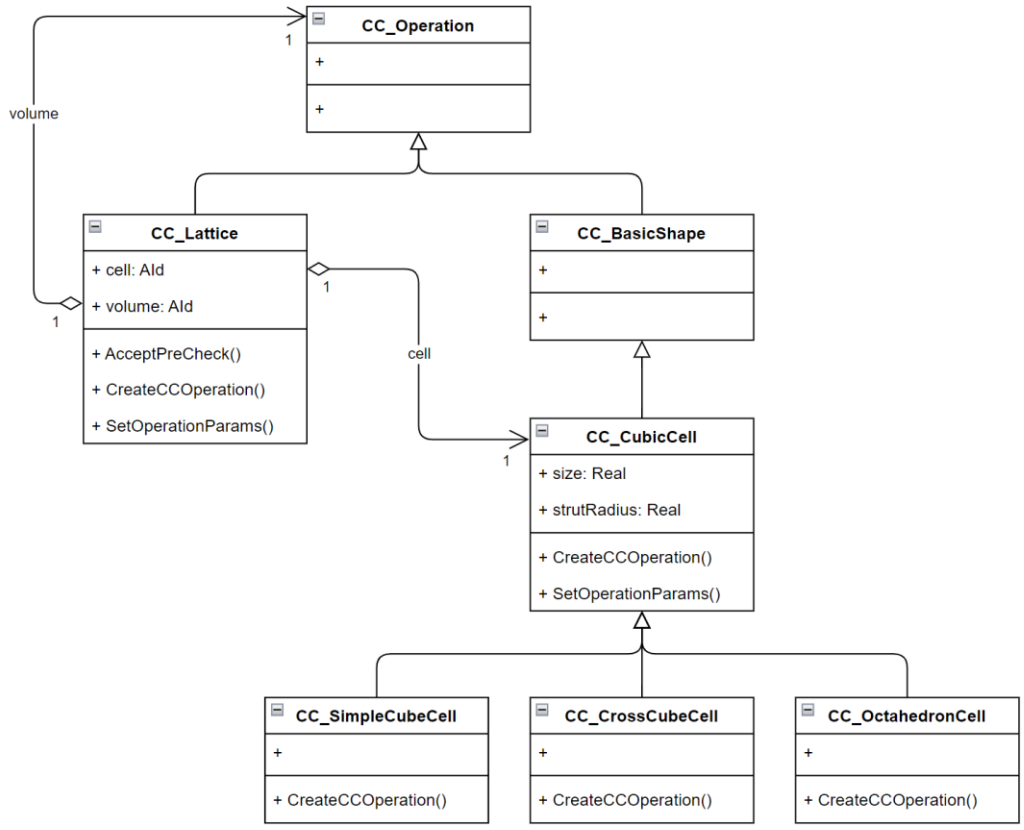

Figure 5-1. Class Diagram

Unit Cell

Classes responsible for unit cells are derived from CC_BasicShape. CC_CubicCell defines the cubic form of a cell, it has two real members, one for the size of the cube and one for the radius of the strut. All children of CC_CubicCell will need those member to construct the struts.

CC_CubicCell overrides SetOperationParams(). The implementation is as follows:

```
1. PAR params;          //[references, size, strutRadius]
2. SetReferences(params[0]);
3. SetMemberExprOrVal(size,params[1]);
4. SetMemberExprOrVal(strutRadius,params[2]);
5. OBJ_Recalc(FALSE,FALSE);
6. RETURN;
```

For test purpose, CreateCCOperation() is also overridden. The simple implementation is as follows:

```
1. VAR cubicCell, success;
2.
3. success = TRUE;
4.
5. cubicCell = CADH_CreateBoxNew(VOID, size, size, size, SELF);
```

```
6.  OBJ_MoveEntity(cubicCell, {-size/2, -size/2, -size/2});
7.
8.  IF ISVOID(cubicCell) THEN
9.            OBJ_ErrorMessage("Failed to create cubic cell",2);
10.           success = FALSE;
11. ENDIF
12.
13. SetResultSolids(cubicCell);
14. SetTransformation();
15.
16. RETURN success;
```

CC_SimpleCubeCell creates struts along all edges of the base cube. It overrides only CreateCCOperation(). Following is the implementation:

```
1.  VAR bar, box, cubeCell, d, success, tmpStrut, tmpStrut2;
2.
3.  success = TRUE;
4.
5.  d = strutRadius * 2;
6.  box = CADH_CreateBoxNew(VOID, size, size, size, SELF);
7.
8.  bar = CADH_CreateConeNew(VOID, size, d, d, SELF);
9.  tmpStrut = CADH_CreateConeNew(VOID, size, d, d, SELF);
10. OBJ_MoveEntity(tmpStrut, {size, 0, 0});
11. bar = CADH_AddSolidNew(bar, tmpStrut, SELF);
12. tmpStrut = CADH_CreateConeNew(VOID, size, d, d, SELF);
13. OBJ_MoveEntity(tmpStrut, {size, size, 0});
14. bar = CADH_AddSolidNew(bar, tmpStrut, SELF);
15. tmpStrut = CADH_CreateConeNew(VOID, size, d, d, SELF);
16. OBJ_MoveEntity(tmpStrut, {0, size, 0});
17. bar = CADH_AddSolidNew(bar, tmpStrut, SELF);
18.
19. tmpStrut = CADH_CopySolidNew(bar, NULLID, SELF);
20. OBJ_RotateEntity(tmpStrut, {-90g, 0, 0});
21. OBJ_MoveEntity(tmpStrut, {0, 0, size});
22.
23. tmpStrut2 = CADH_CopySolidNew(tmpStrut, NULLID, SELF);
24. OBJ_RotateEntity(tmpStrut2, {0, 0, 90g});
25. OBJ_MoveEntity(tmpStrut2, {size, 0, 0});
26.
27. bar = CADH_AddSolidNew(bar, tmpStrut, SELF);
28. bar = CADH_AddSolidNew(bar, tmpStrut2, SELF);
29.
30. cubeCell = CADH_IntersectSolidNew(box, bar, SELF);
31. OBJ_MoveEntity(cubeCell, {-size/2, -size/2, -size/2});
32.
33. IF ISVOID(cubeCell) THEN
34.           OBJ_ErrorMessage("Failed to create cubic cell",2);
35.           success = FALSE;
36. ENDIF
```

```
37.
38. SetResultSolids(cubeCell);
39. SetTransformation();
40.
41. RETURN success;
```

CC_CrossCubeCell creates struts between the centers of opposite faces of the base cube. It creates 4 struts in the middle of the cube. It overrides only CreateCCOperation(). Following is the implementation:

```
1.  VAR crossCell, d, success, tmpStrut, tmpStrut2;
2.
3.  success = TRUE;
4.
5.  d = strutRadius * 2;
6.  crossCell = CADH_CreateConeNew(VOID, size, d, d, SELF);
7.  OBJ_MoveEntity(crossCell, {size/2, size/2, 0});
8.
9.  tmpStrut2 = CADH_CreateConeNew(VOID, size, d, d, SELF);
10. OBJ_RotateEntity(tmpStrut2, {0, 90g, 0});
11. OBJ_MoveEntity(tmpStrut2, {0, size/2, size/2});
12. crossCell = CADH_AddSolidNew(crossCell, tmpStrut2, SELF);
13.
14. tmpStrut = CADH_CreateConeNew(VOID, size, d, d, SELF);
15. OBJ_RotateEntity(tmpStrut, {-90g, 0, 0});
16. OBJ_MoveEntity(tmpStrut, {size/2, 0, size/2});
17. crossCell = CADH_AddSolidNew(crossCell, tmpStrut, SELF);
18. OBJ_MoveEntity(crossCell, {-size/2, -size/2, -size/2});
19.
20. IF ISVOID(crossCell) THEN
21.           OBJ_ErrorMessage("Failed to create cubic cell",2);
22.           success = FALSE;
23. ENDIF
24.
25. SetResultSolids(crossCell);
26. SetTransformation();
27. RETURN success;
```

CC_OctahedronCell creates struts between the centers of nearby faces of the base cube. It creates 12 struts or respectively 8 faces. It overrides only CreateCCOperation(). Following is the implementation:

```
1.  VAR bar, box, octaCell, d, l, success, tmpStrut, tmpStrut2, a, b;
2.
3.  success = TRUE;
4.
5.  d = strutRadius * 2;
6.  l = sqrt((size*size/4) + (size*size/4));
7.  box = CADH_CreateBoxNew(VOID, size, size, size, SELF);
8.
```

```
bar = CADH_CreateConeNew(VOID, l, d, d, SELF);
OBJ_RotateEntity(bar, {45g, 0, 0});
tmpStrut = CADH_CreateConeNew(VOID, l, d, d, SELF);
OBJ_RotateEntity(tmpStrut, {-45g, 0, 0});
bar = CADH_AddSolidNew(bar, tmpStrut, SELF);

tmpStrut = CADH_CopySolidNew(bar, NULLID, SELF);
OBJ_RotateEntity(tmpStrut, {180g, 0, 0});
OBJ_MoveEntity(tmpStrut, {0, 0, size});
bar = CADH_AddSolidNew(bar, tmpStrut, SELF);

tmpStrut = CADH_CopySolidNew(bar, NULLID, SELF);
OBJ_RotateEntity(tmpStrut, {0, 0, 90g});

tmpStrut2 = CADH_CopySolidNew(bar, NULLID, SELF);
OBJ_RotateEntity(tmpStrut2, {0, 90g, 0});
OBJ_MoveEntity(tmpStrut2, {0, 0, size/2});

bar = CADH_AddSolidNew(bar, tmpStrut, SELF);
OBJ_MoveEntity(bar, {size/2, 0, 0});
bar = CADH_AddSolidNew(bar, tmpStrut2, SELF);
OBJ_MoveEntity(bar, {0, size/2, 0});

octaCell = CADH_IntersectSolidNew(box, bar, SELF);
OBJ_MoveEntity(octaCell, {-size/2, -size/2, -size/2});

IF ISVOID(octaCell) THEN
          OBJ_ErrorMessage("Failed to create cubic cell",2);
          success = FALSE;
ENDIF

SetResultSolids(octaCell);
SetTransformation();
RETURN success;
```

Currently, it is not required to add spheres at the joints nor at the ends of each strut because the unit cell maintain its form in the arrangement.

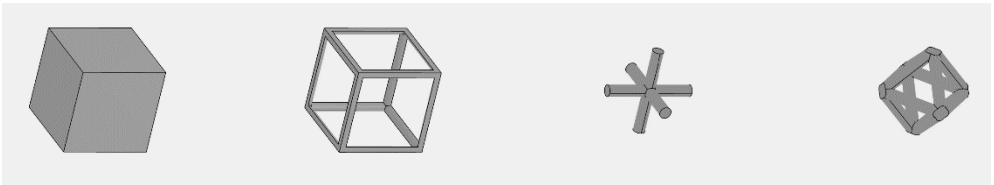

Figure 5-2. Unit Cells

Figure 5-2 shows current unit cells, from left to right: cubic (base cube), simple-cube, cross-cube, and octahedron. The generated tree structure is shown in Figure 5-3 below.

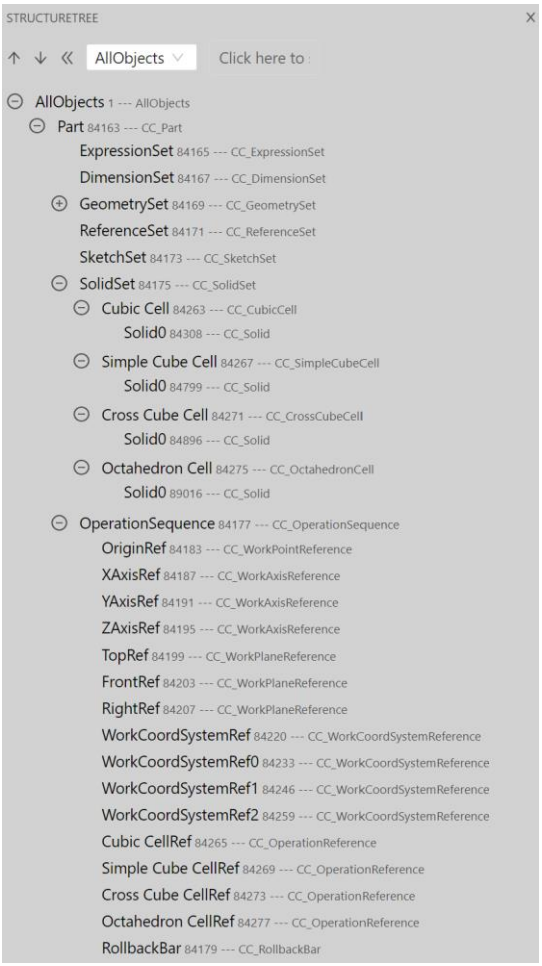

Figure 5-3. ClassCAD Tree

All created unit cells will be added into the solid set and their operation references will also be added into the operation sequence respectively.

Lattice Structure

CC_Lattice is the class responsible for the generation of the lattice structure. For the non-conformal periodic lattice structure, the bounding box of the lattice volume will be used as the boundary for unit cell arrangement in all three directions. The start of arrangement will always be from the center of the bounding box. The orientation of the unit cell will be maintained. Thus, the arrangement will depend on this orientation.

The implementation of CreateCCOperation() in CC_Lattice is as follows:

```
1.  VAR solidVolume, bbVolume, midPoint, csysVol, countx, county, countz, x, y, z, body,
2.  solidCell, csysCell, bbNew, bbCell, size, latticeCell, lineCell, planeCell, tmpCell, id;
3.
4.  size = cell.size;
5.  body = volume.OBJ_GetChildren(2, "CC_Solid");
6.  volume = body[0];
7.  body = cell.OBJ_GetChildren(2, "CC_Solid");
8.  cell = body[0];
9.
```

```
csysVol = volume.OBJ_GetCoordSystem();
csysCell = cell.OBJ_GetCoordSystem();

solidVolume = volume.ConsumeEntities();
solidVolume = solidVolume[0];
bbVolume = OBJ_GetGeomExtents(solidVolume);
midPoint = bbVolume[0] + (bbVolume[1] - bbVolume[0])/2;

cell.OBJ_SetCoordSystem(midPoint, csysCell[1], csysCell[2]);
solidCell = cell.ConsumeEntities();
solidCell = solidCell[0];

bbNew = cell.OBJ_GetGeomExtents(solidVolume);

countx = bbNew[1]:x / size;
latticeCell = CADH_CopySolidNew(solidCell,NULLID,SELF);
cell.OBJ_SetCoordSystem(csysCell[0], csysCell[1], csysCell[2]);
FOR x = 1 TO countx DO
          tmpCell = CADH_CopySolidNew(solidCell,NULLID,SELF);
          OBJ_MoveEntity(tmpCell, x*size*csysCell[1]);
          latticeCell = CADH_AddSolidNew(latticeCell, tmpCell, SELF);
          tmpCell = CADH_CopySolidNew(solidCell,NULLID,SELF);
          OBJ_MoveEntity(tmpCell, -x*size*csysCell[1]);
          latticeCell = CADH_AddSolidNew(latticeCell, tmpCell, SELF);
NEXT

county = bbNew[1]:y / size;
lineCell = CADH_CopySolidNew(latticeCell,NULLID,SELF);
FOR y = 1 TO county DO
          tmpCell = CADH_CopySolidNew(lineCell,NULLID,SELF);
          OBJ_MoveEntity(tmpCell, y*size*csysCell[2]);
          latticeCell = CADH_AddSolidNew(latticeCell, tmpCell, SELF);
          tmpCell = CADH_CopySolidNew(lineCell,NULLID,SELF);
          OBJ_MoveEntity(tmpCell, -y*size*csysCell[2]);
          latticeCell = CADH_AddSolidNew(latticeCell, tmpCell, SELF);
NEXT
CADH_EraseEntity(lineCell);

countz = bbNew[1]:z / size;
planeCell = CADH_CopySolidNew(latticeCell,NULLID,SELF);
FOR z = 1 TO countz DO
          tmpCell = CADH_CopySolidNew(planeCell,NULLID,SELF);
          OBJ_MoveEntity(tmpCell, z*size*csysCell[3]);
          latticeCell = CADH_AddSolidNew(latticeCell, tmpCell, SELF);
          tmpCell = CADH_CopySolidNew(planeCell,NULLID,SELF);
          OBJ_MoveEntity(tmpCell, -z*size*csysCell[3]);
          latticeCell = CADH_AddSolidNew(latticeCell, tmpCell, SELF);
NEXT
CADH_EraseEntity(planeCell);

id = CADH_IntersectSolidNew(solidVolume, latticeCell, SELF);
```

```
61. SetResultSolids(id);
62. RETURN;
```

Test Result

Figure 5-4 shows the basic shape which will be used as lattice volumes. Each volume is separate from the others.

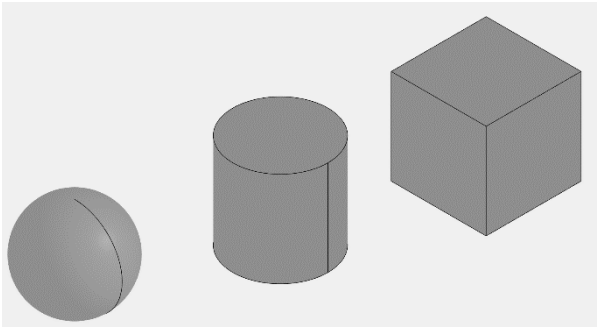

Figure 5-4. Lattice Volumes

Firstly, the lattice structure generation using CC_CubicCell is tested. The unit cell is rotated 30 degrees around the y-axis. For each lattice volume, the generation of lattice structure starts from the center of the volume. Figure 5-5 shows the results.

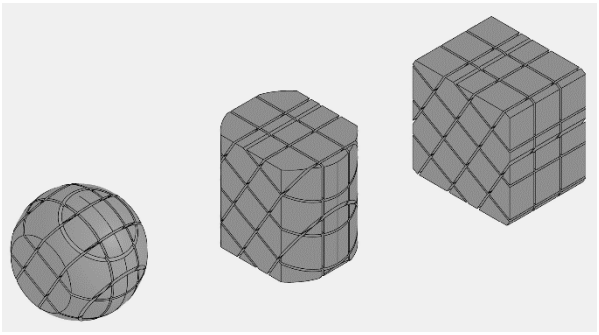

Figure 5-5. Unit Cell Array

Next is CC_SimpleCubeCell. The unit cell is used as it is. Figure 5-6 shows the generated lattice structure on each lattice volume.

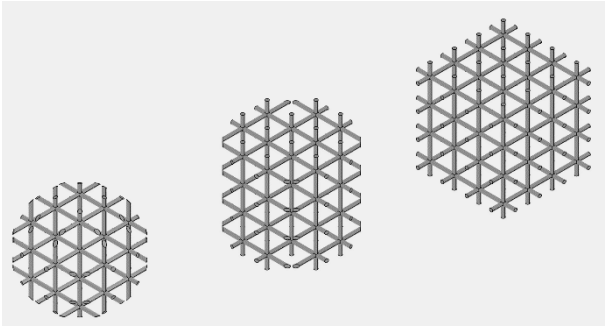

Figure 5-6. Simple Cube

CC_CrossCubeCell is not shown in this document, but it looks basically the same as CC_SimpleCubeCell, but with half-size translation along all axis. Lattice structure using CC_OctahedronCell is shown by Figure 5-7.

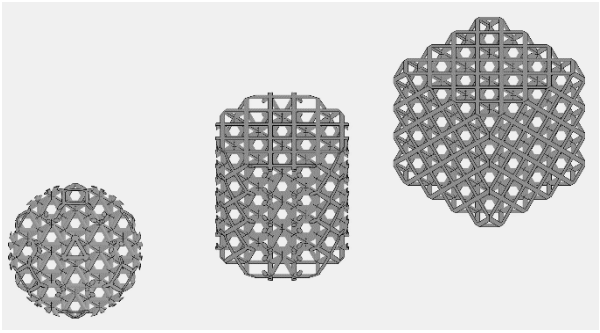

Figure 5-7. Octahedron

The lattice structure generation can also be applied on complex lattice volume. Figure 5-8 shows the generation of lattice structure using CC_OctahedronCell on a lattice volume which created from boolean union of two volumes.

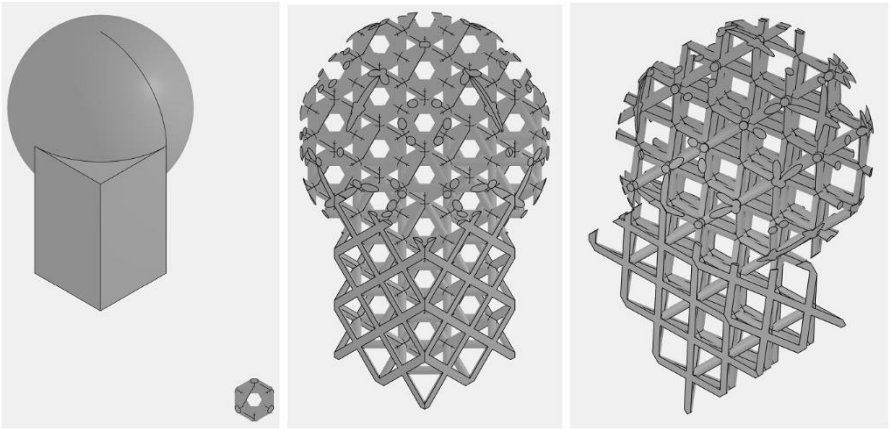

Figure 5-8. Lattice on Union

The generation of non-conformal periodic lattice structure using existing ClassCAD functions is possible. However, the processing time is still a problem. During the above tests, from top to bottom, the increase of the processing time is very noticeable.

5.2.2 PythonOCC

For the second case where mesh is needed to generate lattice structure, access to the 3D modeling kernel is important. Considering the availability and accessibility, Open CASCADE (OCCT) is selected as an alternative. Furthermore, because a Python-wrapper for OCCT exists and the ease of use of Python for research and prototyping, PythonOCC is used for further experimentation.

Development Environment

The main IDE for Python programming used in this project is Microsoft Visual Studio Code. There are only two relevant extensions used in this project, namely Python and Pylance.

To enable the development using PythonOCC, following steps are necessary:

1. Download and install Anaconda Distribution.
2. Using the Anaconda prompt, create specific conda environment for PythonOCC. To create environment called "pythonoccenv" the command is:

   ```
   conda create -n pythonoccenv python=3.8
   ```

   This also specify the use of Python version 3.8.
3. Activate the newly created environment using the command:

   ```
   conda activate pythonoccenv
   ```
4. Install the pythonocc-core package using the command:

   ```
   conda install -c conda-forge pythonocc-core
   ```
5. To test the installation, following command can be run under the active environment containing PythonOCC:

   ```
   python -c "import OCC; print(OCC.VERSION)"
   ```

   If PythonOCC version number is printed, the installation is successful.

Further important packages that need to be install in the same Python environment are:

- PyQT5[6], to enable the desktop user interface for PythonOCC.
- Meshio[7], to help mesh data processing in Python.

Additional tool to generate mesh is Gmsh[8]. This tool can be installed system-wide on the computer.

A different computer was forced to be used in this stage, because the PythonOCC installation was not optimal.

Implementation

In this stage, mesh data is important. For both conformal and non-conformal non-periodic lattice structures, any mesh can be used. However, for conformal periodic lattice structure, specific mesh is needed, because suitable unit cell must be mapped onto the mesh.

As an example, for the implementation purpose, a mesh file of a cube has been generated using Gmsh. This file (**box_05_1.msh**) contains hexahedral mesh and it will be used from now on.

Strut

Now the strut must have extra spheres at both ends, because the full joints cannot be guaranteed to always exist. The implementation for the Strut class is as follows:

```
1. from OCC.Core import BRepBuilderAPI
```

[6] https://pypi.org/project/PyQt5/
[7] https://github.com/nschloe/meshio
[8] https://gmsh.info/

```
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeCylinder, BRepPrimAPI_MakeSphere
from OCC.Core.TopoDS import TopoDS_Shape, TopoDS_Shell
from OCC.Core.gp import gp_Ax2, gp_Ax3, gp_Dir, gp_Pnt, gp_Trsf, gp_Vec

class Strut:
    pt_1 = gp_Pnt(0, 0, 0)
    pt_2 = gp_Pnt(0, 0, 0)
    diameter = 1.0
    origin = gp_Pnt(0, 0, 0)
    normal = gp_Dir(0, 0, 1)
    vect_x = gp_Dir(1, 0, 0)
    axis = gp_Ax2(origin, normal, vect_x)

    def __init__(self, pt_1, pt_2, diameter=1.0) -> None:
        self.diameter = diameter
        self.pt_1 = pt_1
        self.pt_2 = pt_2
        self.height = self.pt_1.Distance(self.pt_2)
        self.origin = pt_1
        self.normal = gp_Dir(gp_Vec(self.pt_1, self.pt_2))
        #print(self.normal.X(),self.normal.Y(), self.normal.Z())
        self.vect_x = self.get_perpendicular_vec(self.normal)
        #print(self.vect_x.X(),self.vect_x.Y(), self.vect_x.Z())
        self.axis = gp_Ax2(self.origin, self.normal, self.vect_x)

    def get_perpendicular_vec(self, vec) -> gp_Vec:
        x, y, z = 1, 1, 1

        if vec.X() > 0 :
            x = 0
        elif vec.Y() > 0 :
            y = 0
        else:
            z = 0

        v = gp_Dir(gp_Vec(x, y, z))
        perp_vec = vec.Crossed(v)

        return perp_vec

    def create(self) -> TopoDS_Shape:
        cyl = BRepPrimAPI_MakeCylinder(self.axis, self.diameter/2, self.height).Shape()
        sphere1 = BRepPrimAPI_MakeSphere(self.pt_1, self.diameter/2).Shape()
        sphere2 = BRepPrimAPI_MakeSphere(self.pt_2, self.diameter/2).Shape()
        strut = BRepAlgoAPI_Fuse(cyl, sphere1).Shape()
        strut = BRepAlgoAPI_Fuse(strut, sphere2).Shape()
        return strut

    def __eq__(self, other):
        tol = 0.01
```

```
        print(other.pt_1, other.pt_2)
        if (self.pt_1.IsEqual(other.pt_1, tol) and self.pt_2.IsEqual(other.pt_2, tol)) or
   (self.pt_1.IsEqual(other.pt_2, tol) and self.pt_2.IsEqual(other.pt_1, tol)):
            return True
        else:
            return False
```

An implementation example for drawing a strut is:

```
strut = Strut(pt_1=gp_Pnt(0,0,0), pt_2=gp_Pnt(100,100,100), diameter=10)
strut = strut.create()
display.DisplayShape(strut, update=True, color='ORANGE')
```

And the result is as follows:

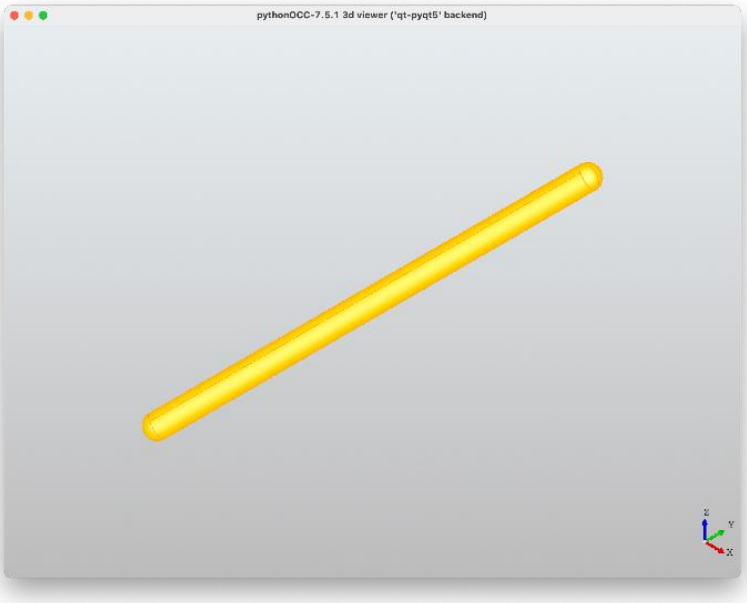

Figure 5-9. Strut

Lattice Structure

In this implementation, only hexahedron cells will be read. There are 8 points on each cell, but most of them belong also to other cells. In the direct conversion from edges to struts, it is important to track the generated struts to avoid copy of struts at the same edges.

Following is the snippet to draw struts on mesh:

```
mesh = meshio.read(
            "box_05_1.msh"
        )
datas = mesh.get_cells_type("hexahedron")
data = datas[0]
points = mesh.points

```

```
strutlist = []

print("Hexahedron numbers: {}".format(len(datas)))

for data in datas:
    for i in range(0,4):
        p1 = data[i]
        p1_ = data[i+4]
        if i == 3:
            p2 = data[0]
            p2_ = data[4]
        else:
            p2 = data[i+1]
            p2_ = data[i+4+1]

        strut = [p1,p2]
        if not Helpers.isPairExists(strutlist, strut):
            strutlist.append(strut)

        strut = [p1_,p2_]
        if not Helpers.isPairExists(strutlist, strut):
            strutlist.append(strut)

        strut = [p1, p1_]
        if not Helpers.isPairExists(strutlist, strut):
            strutlist.append(strut)

print("Strut numbers: {}".format(len(strutlist)))

for strut in strutlist:
    p1 = points[strut[0]]
    p2 = points[strut[1]]
    strutshape = Strut(pt_1=gp_Pnt(p1[0],p1[1],p1[2]), pt_2=gp_Pnt(p2[0],p2[1],p2[2]),
diameter=0.01)
    strutshape = strutshape.create()
    display.DisplayShape(strutshape, update=True, color='ORANGE')
```

Meanwhile, the implementation of Helpers class is:

```
import sys
import os
import numpy as np
import meshio

sys.path.append(os.path.realpath('..' + os.path.sep + 'cells' + os.path.sep))
sys.path.append(os.path.realpath('..' + os.path.sep + 'structures' + os.path.sep))
from cubecell import Cubecell
from strut import Strut
from lattice import Lattice

class Helpers:
```

```
13.     def isPairExists(list, pair)->bool:
14.         exist = False
15.
16.         if ([pair[0], pair[1]] in list) or ([pair[1], pair[0]] in list):
17.             exist = True
18.
19.         return exist
```

Test Result

Although mesh file is provided, the lattice construction as shown in Figure 5-10 below takes quite some time.

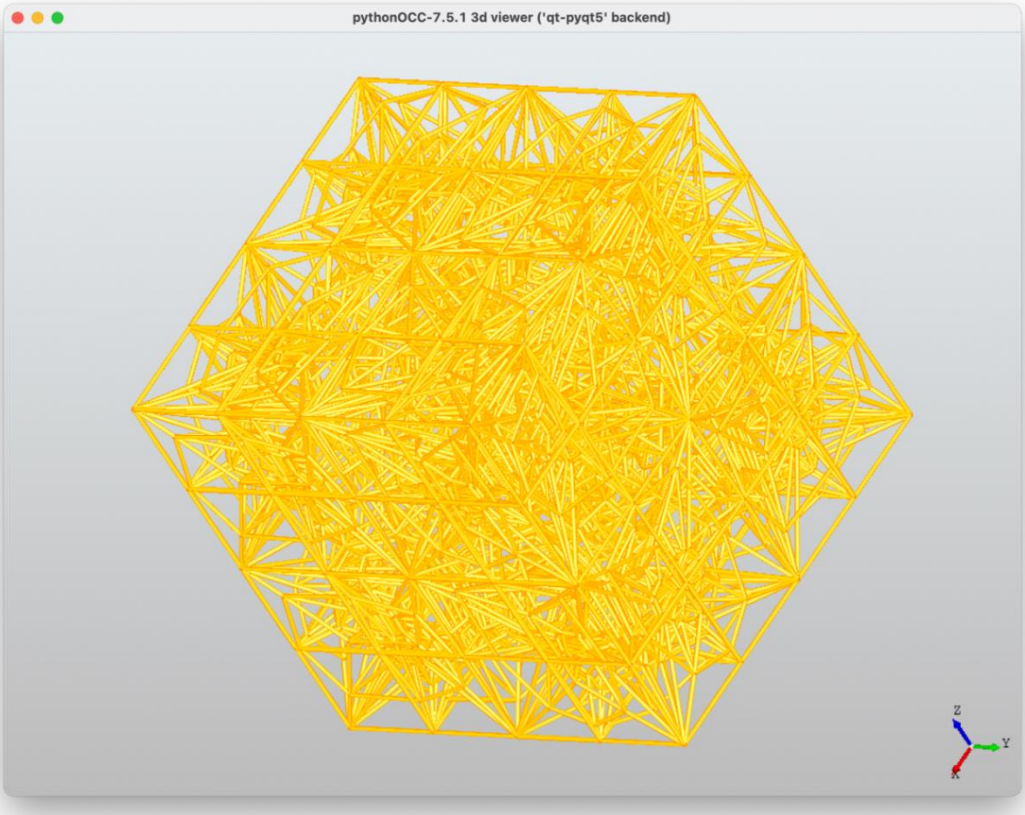

Figure 5-10. Conformal Non-Periodic Lattice

The processing time for all the experiments that have been carried out is not satisfactory.

6 Conclusions

Lattice structure for additive manufacturing is a topic that is quite often studied and researched. The methods to generate the lattice structure which are discussed on referenced papers are similar. Even on the studied application in this project, the steps and procedures reflect the methods from those papers.

Based on the studies and experiments carried out in this project, generating lattice structure can be implemented in ClassCAD. However, the attempt to use the current functionality does not seem good enough to provide some efficient processes. Using basic primitives to create lattice structure is possible, but overhead awaits up front. Work-arounds and other methods need to be explored, especially those related to the third-party services used in ClassCAD.

Open CASCADE has also capabilities that need to be explored further. Using PythonOCC turned out to be quite fun and the possibility to run well on different platforms is staggering.

6.1 Lessons Learned

3d application development requires adequate resources, especially during the test-run. What can be applied may not necessarily work properly. Proper estimation, as well as anticipation and early detection of system failures will provide convenience for subsequent processes.

Working with multiple applications, libraries, and devices requires careful attention to detail. Logs or records are needed, especially regarding version changes or configuration changes. It will be very time consuming if a problem arises that cannot be immediately identified whether it is related to the environment configuration or implementation.

Careful and meticulous planning is necessary, both on the project execution and on the documentation.

6.2 Encountered Problems

Several applications and libraries turned out to be unusable because they are no longer supported.

Sometimes the system seems to be running well but no results are obtained, and sometimes it's the other way around. This still cannot be identified, it might be caused by the implementation or other applications that currently running or hardware or some other reasons.

6.3 Future Work

Further research on lattice structure is still needed. There are so many possibilities in term of generating lattice structure. For the lattice structures that use meshes, it might be necessary to integrate mesh generator into the lattice generator. Furthermore, incorporate the loading and material properties into the construction process.

In term of ClassCAD, study on the underlying technologies and libraries to optimize the process of generating lattice structure. Processing time and memory usage are the main observations.

7 List of Figures

8 References

Botsch, Mario, et al. *Polygon mesh processing*. CRC press, 2010.

Nagesha, B. K., et al. "Review on characterization and impacts of the lattice structure in additive manufacturing." *Materials Today: Proceedings* 21 (2020): 916-919.

Nguyen, Dinh Son, and Frédéric Vignat. "A method to generate lattice structure for additive manufacturing." *2016 IEEE international conference on industrial engineering and engineering management (IEEM)*. IEEE, 2016.

Nguyen, Dinh Son, et al. "Creation of Lattice Structures for Additive Manufacturing in CAD Environment." *2018 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. IEEE, 2018.

Pan, Chen, Yafeng Han, and Jiping Lu. "Design and optimization of lattice structures: A review." *Applied Sciences* 10.18 (2020): 6374.