



Technical University of Munich

Faculty of Electrical Engineering and Information Technology

Chair of Design Automation

## Bachelor's Thesis

Phoon Jia Wen

Supervisor University Professor : Prof. Ulf Schlichtmann

Supervising Assistant : Yushen Zhang

Issue of the Topic : Development of an STL Library for 3D-Printing Design

Submission Data : 24 May 2021

# 1. Table of Contents

1. Table of Contents.....	2
2. Table of Figures.....	3
3. Abstract.....	5
4. Introduction .....	6
5. Background .....	7
5.1. Library.....	7
5.2. STL.....	8
5.2.1. Special Rule for STL .....	10
5.2.2. How STL is Printed.....	12
5.3. Java .....	14
5.4. Microfluidic Chips.....	16
6. UML Class Diagram .....	17
6.1. Point Class.....	18
6.2. Rectangle3D Class.....	19
6.3. Cylinder Class.....	21
6.4. Polygon Class .....	22
6.5. Node Class .....	23
6.6. Solid Class .....	25
6.7. StlWriterUtil Class.....	27
7. STL Model .....	29
7.1. Calcium Lactate with Sodium Alginate.....	29
7.2. Ring Normal .....	32
7.3. Mixer.....	35
8. Conclusion.....	39
9. References .....	40

## 2. Table of Figures

Figure 1: Example of Tessellation [5] .....	8
Figure 2: Tessellated Cube .....	9
Figure 3: Facets [6] .....	9
Figure 5: Vertex Rule [7] .....	10
Figure 6: Orientation Rule [6] .....	11
Figure 7: 3D Cartesian Coordinate System [8] .....	11
Figure 8: Spherical Surface Tessellation [9] .....	13
Figure 9: Microfluidic Chip 1 [14] .....	16
Figure 10: Microfluidic Chip 2 [1] .....	16
Figure 11: Microfluidic Chip 3 [15]] .....	16
Figure 12: UML Class Diagram .....	17
Figure 13: Point Class .....	18
Figure 14: Rectangle3D Class .....	19
Figure 15: Rectangle3D Constructor .....	20
Figure 16: Rectangle3D Allocation .....	20
Figure 17: Cylinder Class .....	21
Figure 18: Polygon Class .....	22
Figure 19: Node Class .....	23
Figure 20: Solid Class .....	25
Figure 21: Shape Demonstration [16] .....	25
Figure 22: Union Combination Solid Shape [16] .....	26
Figure 23: Subtract Combination Solid Shape [16] .....	26
Figure 24: Intersect Combination Solid Shape [16] .....	26
Figure 25: StlWriterUtil Class .....	27
Figure 26: Calcium Lactate with Sodium Alginate Interior .....	29
Figure 27: Calcium Lactate with Sodium Alginate Exterior .....	29
Figure 28: Calcium Lactate with Sodium Alginate Top View -Baseless .....	30
Figure 29: Calcium Lactate with Sodium Alginate Side View 1 - Baseless .....	30
Figure 30: Calcium Lactate with Sodium Alginate Side View 2 - Baseless .....	30
Figure 31: Calcium Lactate with Sodium Alginate Side View 3 - Baseless .....	30
Figure 32: Calcium Lactate with Sodium Alginate Top View with Base .....	32
Figure 33: Calcium Lactate with Sodium Alginate Side View 1 with Base .....	32
Figure 34: Calcium Lactate with Sodium Alginate Side View 2 with Base .....	32
Figure 35: Calcium Lactate with Sodium Alginate Side View 3 with Base .....	32
Figure 36: Ring Normal Interior .....	33
Figure 37: Ring Normal Exterior .....	33
Figure 38: Ring Normal Top View .....	33
Figure 39: Ring Normal Cylinder View .....	33
Figure 40: Ring Normal Interior View 1 .....	33
Figure 41: Ring Normal Interior View 2 .....	33

Figure 42: Mixer Interior .....35  
Figure 43: Mixer Exterior .....35  
Figure 44: Mixer Bottom View - Baseless .....36  
Figure 45: Mixer Top View - Baseless .....36  
Figure 46: Mixer Top View with Base.....38  
Figure 47: Mixer Bottom View with Base .....38  
Figure 48: Mixer interior View 1 with Base.....38  
Figure 49: Mixer interior View 2 with Base.....38

### 3. Abstract

In this bachelor's thesis, we are going to study the development of an STL library for a 3D printing. It would break down into few parts to study the detail of this topic. Firstly, we will talk about the background knowledge of how an STL is formed, what is a library in Java, understand the structure of a library and the target object for our 3D image.

After having some background knowledge, we would go on to focus more on the design. Our aim is to create a 3D image of a microfluidic chip. Hence, it is important to understand the different structure of a microfluidic chip. What are the important things to take note when doing the design if the chip.

We also will have a detailed explanation of the Java code in the form of UML class diagram. It would explain individual classes to allow reader to understand the library structure. Lastly, there would also be some STL models provided to further understand the structure and the use of the library.

## 4. Introduction

Microfluidic chip is commonly use in the biomedical field as well as cell biology research. Microfluidic chip allows to integrate many medical tests with researching on a chip. The reason why it is commonly used in this industry is because the micro channel is very similar to the biological cell. It is similar in term of the characteristic size of the channel. Therefore, simple manipulation of a single cells is easy to permit in microfluidic chip and it also allows quick medication changes. [1]

As many designers are designing their microfluidic chip with scratch. Hence, a lot of times needs to spend on finding the algorithm and the structure behind all the different shapes and design. It became more difficult as developer need to consider many issues while trying to do the design of the chips. Therefore, in this project, I am going to design an STL library that is mainly focus on designing a microfluidic chip. I am going to use java program to create this STL library. In the library, users would be able to use functions such as adding shapes and inserting the dimensions of the shapes in x-plane, y-plane, and z-plane for them to build their designed microfluidic chip.

For the easy usage of the library, there would explanation provided for user whenever they are inserting or using the functions. This would allow them to understand better how to use this library.

In the UML class diagram, it would show the mind map of the formation of the library and detailed functions and methods that is used. There are total of seven classes, which are Point class, Rectangle3D class, Cylinder class, Node class, Polygon class, Solid class and StlWriterUtil class. Every class is used for different purposes.

The motive of this bachelor's thesis is to provide simplification for developer to design the object. Hence, we aim to create code that is easy to understand as well as easy to use.

## 5. Background

Before we start on the design of the library, we need to analyse the topic so we would know how to start on our design. From looking at the topic, “Development of an STL Library for 3D-Printing Design”, firstly, we would need to understand what is a library. We also need to know what is an STL and how the 3D image is formed with it. As we are using Java programming to create this library, research are also needs to be done before starting the implementation.

### 5.1. Library

A library is an assortment of non-volatile assets utilized by computer programs in software engineering. There is many information that can be include in a library. Information such as documentation, help data, configuration data, pre-written code, message templates, classes, values, type specifications or subroutines.

All the information in the library is also known as a collection of implementations of behavior. The library would show an all-around characterized interface where the behavior is used. Furthermore, the implementation of behavior in a library is created to allow user to use for multiple independent programs. For instance, a high-level program would usually use a library to call their system instead of implementing the system and call it repeatedly. This is how library is being organized. It can be used in multiple programs with no connection to every programs. This would benefit user who is creating a large program as it can acquire a various leveled idea when a program develops huge.

Another advantage of using a library is that user do not need to know the internal detail of the library. Instead, user only need to know the interface as there would be distinctive features that a library is coordinated for the reasons for being reused in multiple programs independently. This saves a lot of time because user does not have to write the program from scratch. [2]

A Java Class Library (JCL) is a set of progressively loadable libraries that Java Virtual Machine (JVM) languages can call at run time. The application cannot depend on any of the stage local libraries because Java Platform is not reliant on a specific operating system. A Java Platform gives a thorough arrangement of standard class libraries, containing the functions common to modern operating systems.

There are three purposes that JCL serve within the JVM. Firstly, JCL provide useful facilities such as container classes and regular expression processing for programmer just like other standard code libraries. An abstract interface such as file access and network access would also be provided by library to tasks on the hardware and operating system that would usually rely heavily on. Moreover, some of the features might not be supported by Java application. Hence,

the library can try to copy those features or have a steady method to check for the presence of a specific feature.

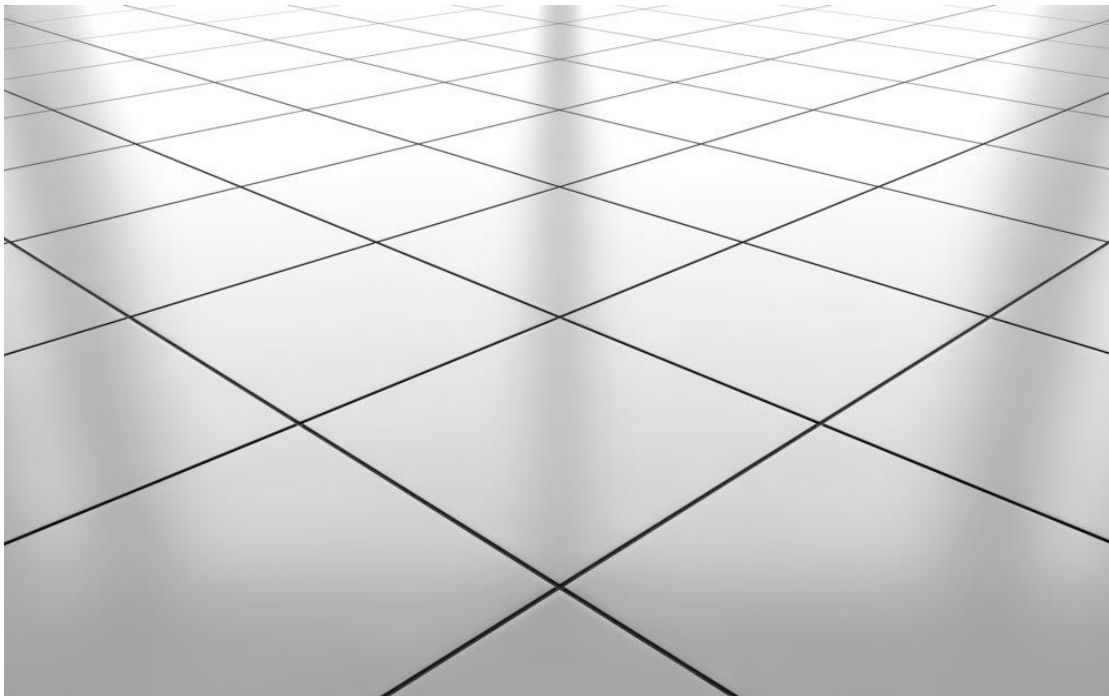
JCL is mostly written in Java, except some parts where we need direct access to the operating system. [3]While in this project, I would need to convert the Java to an STL format.

## 5.2. STL

STL stand for Stereo Lithography, it is a file format to create a 3D system. Many software packages are being supported by STL. For instance, it is commonly used in 3D printing, computer aided manufacturing as well as rapid prototyping. This file format describes an object of its surface geometry in three dimensions. An STL format have two kind of representations which are ASCII and binary while in this project, we would only focus on doing the ASCII representation. [4]

### Tessellation

Now that we have a basic understanding of what is an STL file, we have come to another question which is how a 3D model is being store in an STL file format? An STL file format is focusing on the 3D object of its surface geometry. To form the 3D object, we would use a concept which called “tessellation”. Tessellation refers to a process of tiling a surface with at least one geometric shape in the 3D model with no gaps and overlaps. A good life example of tessellation would be a tile floor.



*Figure 1: Example of Tessellation [5]*



Hence, an STL also known as Standard Tessellation Language. The idea of tessellation is using tiny triangles (also known as facets) to model the 3D object in two-dimensional outer surface. After the modelling, the information of the facet would store in the file. For instance, a 3D cube of its surface could be approximated with twelve triangles. This can be shown in Figure 2 below.

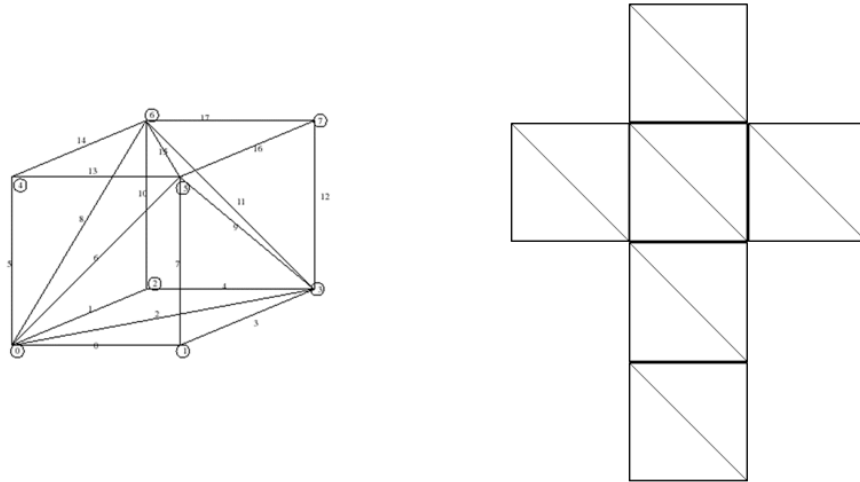


Figure 2: Tessellated Cube

The more complex the surface, the more triangles produced. Also, the larger the STL file, the more triangles placed on the surface of the model. [6]

## Facets

There is some information we need to look for about the facets in an STL file. The first thing we are looking for would be the coordinates of the vertices. The coordinates of the vertices must be in x-, y- and z-dimensional. Second, would be the unit normal vector to the triangles. The normal vector of the facet should be pointing outwards with respect to the 3D model. [6]

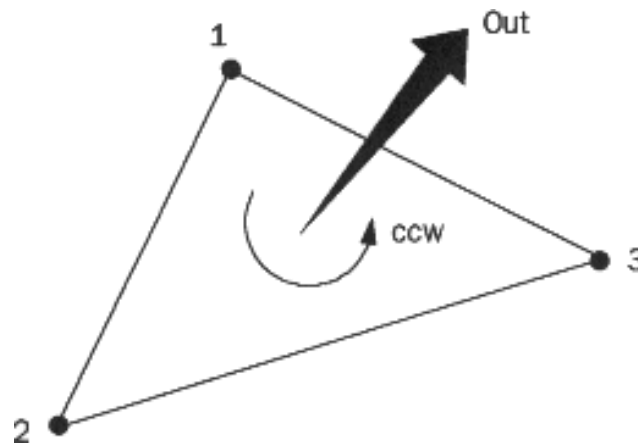


Figure 3: Facets [6]

### 5.2.1. Special Rule for STL

There are also some special rules that we should take note when designing the facets in STL file format. Rules such as vertex rule, orientation rule, all positive octant rule as well as triangle sorting rule.

#### Vertex Rule

In the vertex rule, it says that “The vertex rule states that each triangle must share two vertices with its neighboring triangles”.

This rule is applying to the 3D object when we are designing the tessellation of its surface. As the object’s surface would be built with many tiny triangles, to form an STL format, all the triangles on the object’s surface must be connected such that 3 points in the triangle would have 2 points sharing the same position (or coordinate) with the neighboring triangle.

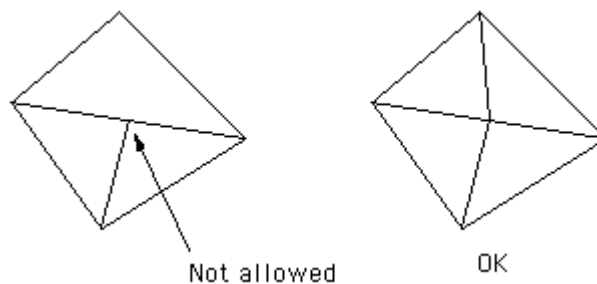


Figure 4: Vertex Rule [7]

Figure 4 shows an example of how vertex rule is being applied. As we can see from the left image, one of the triangles is only sharing one pint each, to the other two triangles. This has violated the vertex rule and it is also not a valid tessellation. While for the right image, all the triangles are sharing exactly 2 vertices to other triangles. This shows the correct vertex rule is applied and how tessellation should be formed. [6]

#### Orientation rule

In the orientation rule, it says that “The orientation of the facet (i.e., which way is “in” the 3D object and which way is “out”) must be specified in two ways”.

There are two parts about this orientation rule. One would be the normal of the triangle must be in the direction of pointing outwards. Secondly, the order of the 3 vertices in the triangle must be in the counterclockwise direction when looking from the outside of the object. We also can check the orientation rule by using the right-hand rule.

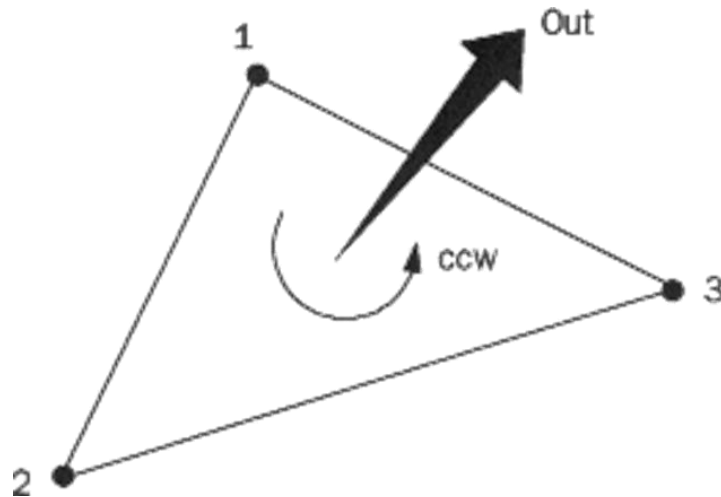


Figure 5: Orientation Rule [6]

One advantage of having orientation rule is that it would help to prevent the corruption occurs and maintain the consistency of the data. By having the consistent orientation of the vertices and normal, a software can verify easily how the object is formed in STL format. [6]

### All Positive Octant Rule

In all positive octant rule, it says that “The coordinates of the triangle vertices must all be positive”.

From the name of this rule, we can tell that the vertices in all triangles must live in all positive octant. This is applied to the 3D Cartesian coordinate system.

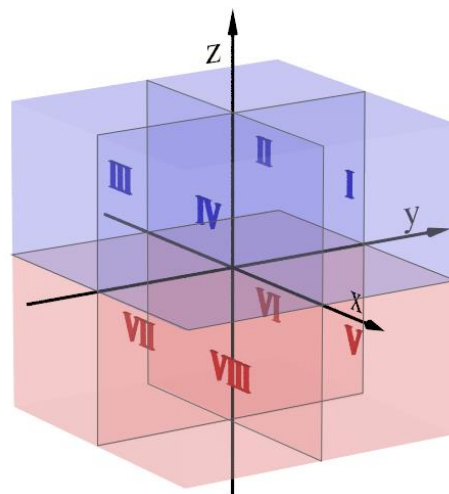


Figure 6: 3D Cartesian Coordinate System [8]

From Figure 6, octant I represent all positive octant. The reason why we have this rule when building STL file is to save space. [6]

## Triangle Sorting Rule

Triangle sorting rule, it says that “The triangles appear in ascending z-value order”.

By putting the triangles in ascending order in z direction, it allows the slicers to process faster in a 3D model. This is recommended while designing the 3D model, however, it is not compulsory to apply this rule while designing and the 3D model can still be generated even we are not using the triangle sorting rule. [6]

### 5.2.2. How STL is Printed

After knowing all the rules that are needed to create an STL file, now we need to understand how it is printed. In STL file, a 3D printing needs to be generated in designed slicer. A slicer is a software that can be used in 3D printer. It would help to create an object with the printing instructions that is converted from digital 3D models.

A slicer will cut the STL file into many slices of flat horizontal layers depending on the amount that is set by user. Hence, it could be in a range of hundreds to thousands of slices. It also will calculate the time taken to create and how much material is needed to extrude from your printer.

After collecting all the information, it would then bundle up and formed a GCode file. This is a native language for the 3D printer. [6]

## Optimization

As we know that STL file format is created by approximate the surface of the 3D model in triangles, hence, we will never get a perfect shape for a rounded 3D model. One reason is the facets will create coarseness to the 3D model. This would be a problem as the 3D printer will print exactly how a coarseness model is created in the STL file.

To solve this issue, we can make the triangles as small as possible to achieve the finest of the 3D model. This is because the smaller the triangle is, the better the approximation, hence it would result in a good quality 3D model. [6]

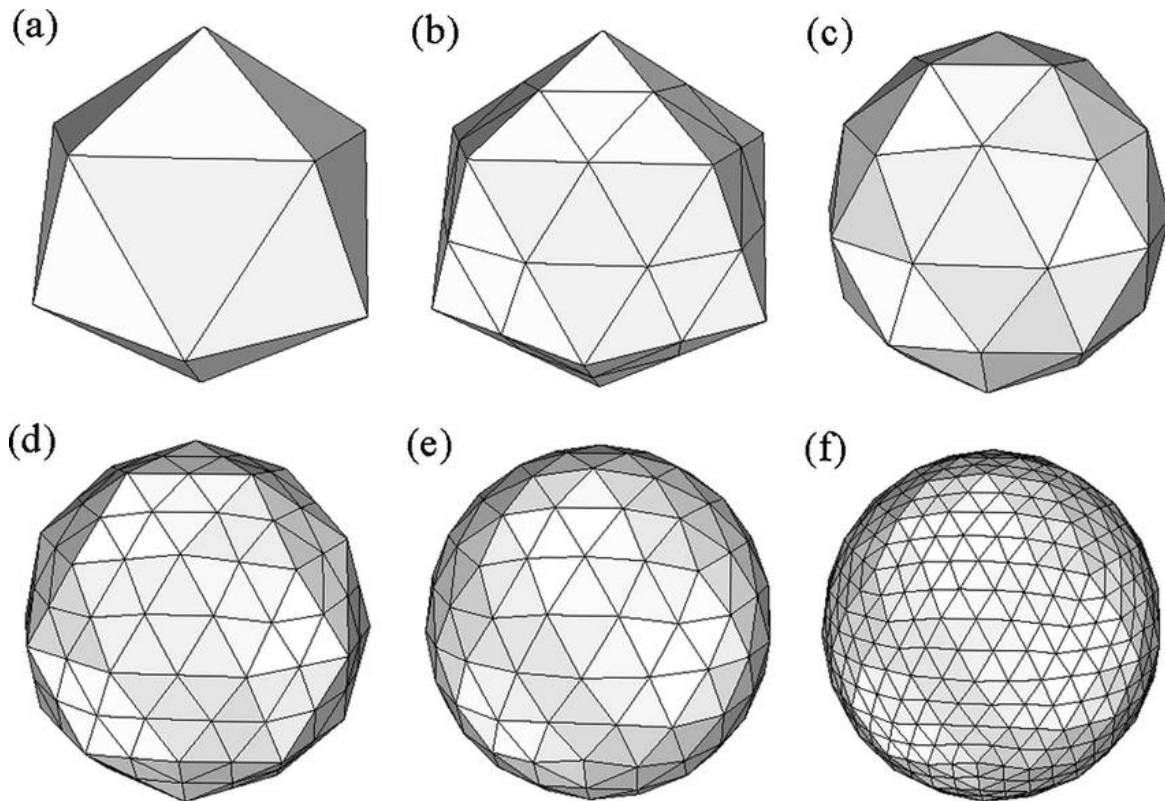


Figure 7: Spherical Surface Tessellation [9]

As we can see from the figure 7 above, it shows how the smaller triangles can tessellate into a better spherical surface.

However, there is a limitation on tessellating small triangles. As we can see from figure 7, we can tell that if the size of the triangle decreases, it would require more triangles to cover up the entire surface of the sphere. This will result in a large STL file generated and the 3D printer might not be able to support a gigantic STL file. Therefore, it is important to get the right balance of the quality of the print and the size of the file.

## ASCII and Binary

There are two different ways to store the information in STL file, one is ASCII encoding and another one would be binary encoding.

ASCII STL files are relatively easy to read and check. It also allows debugging for user who want to perform manual inspection. Therefore, it is good for those who want to do debugging as it can be read easily.

Binary STL files are smaller as compared to ASCII STL files and it is also easy to share. Hence, binary encoding is always recommended to use for 3D printing. [10]

## 5.3. Java

In this project, I am going to use Java to design my STL library. What is Java? Java is an object-oriented programming language and a class-based programming language that is designed to have lesser implementation dependencies. It is also known as general-purpose programming language that allow programmer to compile their Java code and run anywhere that is supported by Java. Hence, it is wisely use in programming world because of the reliability, security and fast. [11]

### **OOP Concepts in Java**

Before we start on the coding, we need to understand a few concepts that is very useful while writing the code in Java. OOP stands for Object-Oriented Programming. So now, what is OOP concept in Java? It is a concept that allow developer to create their variables and working methods, then without compromising the security, re-utilize part or all of them. There is total four main OOP concept in Java, which are abstraction, encapsulation, inheritance, and polymorphism. [12]

#### **Abstraction**

Abstraction refers to a complex code and data that is represented using a simple thing. For instance, the representation of simple things in Java such as variables, objects, and classes. The advantage of abstraction is to avoid rehashing the same work on different occasions. With this, programmer would create different classes or different types of objects. A simple example would be he/she can create a class of variable: address. Each addresses object would have name, street, city, and postal code. The object can be employee addresses, customer addresses or supplier addresses. [12]

#### **Encapsulation**

Encapsulation refers to storing the data or code in a private class and provide a public method to allow accessing. In this way, it helps to protect the information safe within the class as it acts as a protective barrier for the code and data. This concept is useful as it allows re-using of the functionality without endangering the security. It also saves a great deal of time. A simple example would be programmer uses the same variable that is assigned in private and use it in different functions. [12]

#### **Inheritance**

Inheritance refers to sharing the attribute of the existing classes in the new class created. An inheritance class is also act as a subclass or child class. This allows us to expand on the past

work without wasting time. We would use the word “extends” when performing inheritance from the parent class to the subclass. [12]

## Polymorphism

Polymorphism refers to different things in various setting that is using the same word to represent. There are two types of polymorphism which are overloading and overriding.

Overloading is a compile time, and it refers to the same code but represent different meanings. For instance, a class overloading is shown below:

```
class overloading{
    void Rectangle(){
        System.out.println("No parameter");
    }
    void Rectangle(double x, double y){
        System.out.println("x:" + x + "y:" + y );
    }
    void Rectangle(double x, double y, double z){
        System.out.println("x:" + x + "y:" + y + "z:" + z );
    }
    public static void main(String args[]){
        overloading ol = new overloading();
        ol.Rectangle();
        ol.Rectangle(10, 20);
        ol.Rectangle(10, 20, 30);
    }
}
```

As we can see from the example, the same Rectangle method is being use multiple time with different parameters that passes through.

Overriding is a run time and it refers to the value of the variable is assigned to different meanings. For instance, a class overriding is shown below:

```
class overriding{
    public void move(){
        System.out.println("Animals can move");
    }
}
class Cat extends overriding{
    public void move(){
        System.out.println("Cat can move");
    }
    public static void main(String args[]){
        Cat c = new Cat();
        c.move();
    }
}
```

The outcome would be: “Cat can move” instead of “Animals can move”. This shows the program is overriding the information that is store in the method. [12]

## 5.4. Microfluidic Chips

In this an STL library, I am going to design based on a microfluidic chip. Hence, it is important to understand what and how a microfluidic chip is. What is microfluidic chip? It is a device that use to examine in which micro-channels have been shaped or designed. The channels shaping in the chip are associated together to permit liquids to go through various channels and moving between different places. This network channels are associated through inlet and outlet ports to expose to the outside environment. The chip will allow injection, manage, and remove the fluids (or gases) inactively or effectively. There can be different internal diameter for the channel, the usual range would be between 5 to 500 $\mu\text{m}$ . However, today constructions can be manufactured with sub-micrometer accuracy. The channel network should be explicitly designed for the ideal application and testing. For instance, DNA analysis, lab-on-a-chip, cell culture, microfluidic droplets, cell culture and many more. [13]

Below would be some of the life sample of how a microfluidic chip is look like. The microfluidic chip can have many inlets and outlets and the channel is also can be designed in different shape and size depending on the researcher requirement.

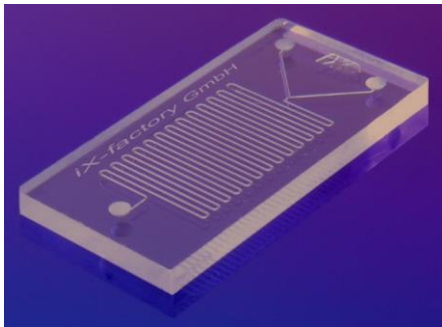


Figure 8: Microfluidic Chip 1 [14]

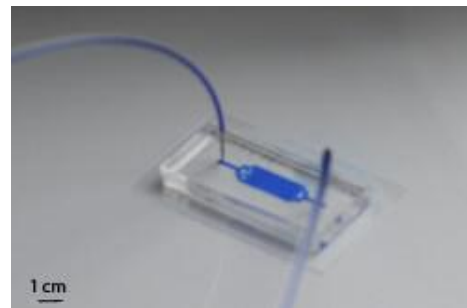


Figure 9: Microfluidic Chip 2 [1]

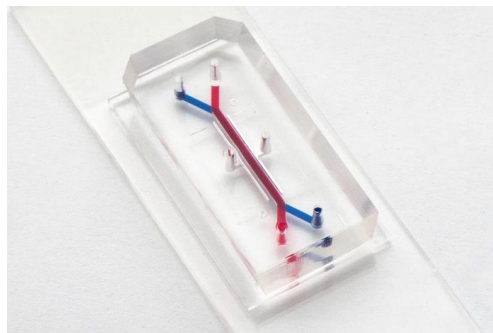


Figure 10: Microfluidic Chip 3 [15]]



## 6. UML Class Diagram

In this session, I will show the UML class diagram of my project. There are total of seven classes. Every classes have their different usages.

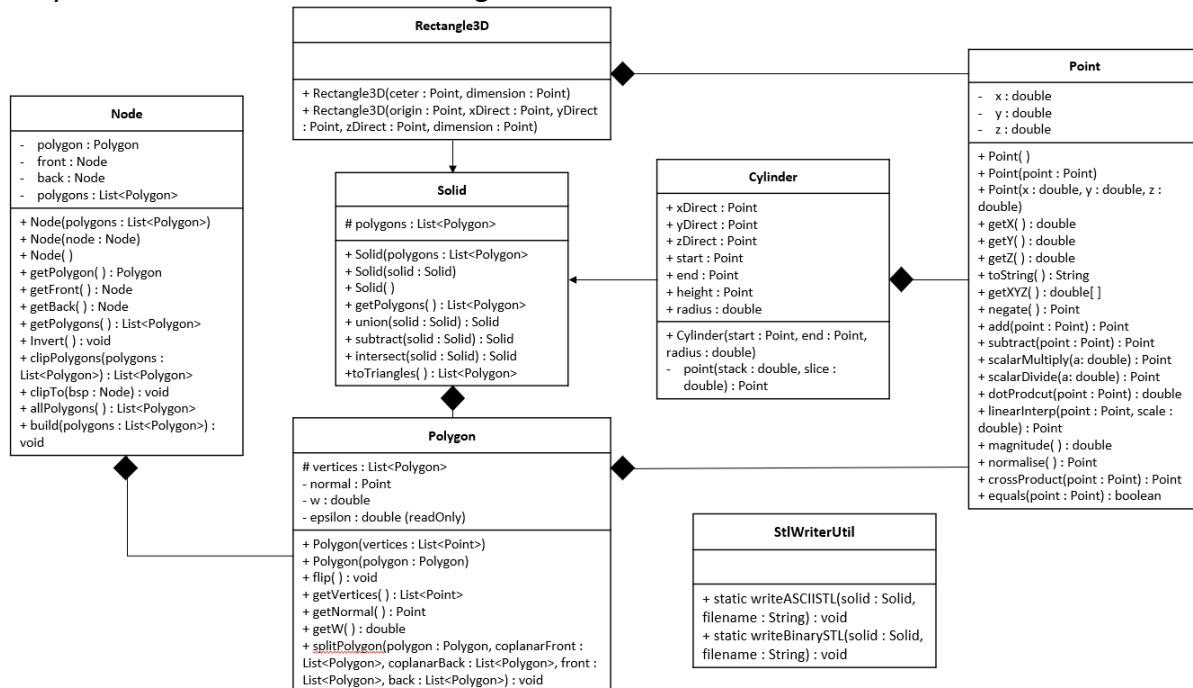


Figure 11: UML Class Diagram

As you can see from the figure above, Point class, Rectangle3D class, Cylinder class, Node class, Polygon class and Solid class are related while StlWriterUtil class is isolated by itself. The StlWriterUtil class is a class to provide conversion from Java to STL file format. Hence, it is isolated by itself.

As for the remaining classes, firstly, we can see that Rectangle3D class, Cylinder class and Polygon class are composited to Point class. Point class is the basic class to generate different points in three-dimensions, without Point class, we cannot draw any shape.

We also can see that Solid class and Node class is composited with polygon class. This is because polygon class is the class to create points on a single plane. While Node class is to tell the location of every points and solid class is a class to combine all the planes together. Hence without polygon class, the two class will not work.

Finally, we can see that Rectangle3D class and Cylinder class are derived from the Solid class to form the 3D object. The detailed explanation of each classes is to clarify further at the session below.

## 6.1. Point Class

Point
- x : double - y : double - z : double
+ Point( ) + Point(point : Point) + Point(x : double, y : double, z : double) + getX( ) : double + getY( ) : double + getZ( ) : double + toString( ) : String + getXYZ( ) : double[ ] + negate( ) : Point + add(point : Point) : Point + subtract(point : Point) : Point + scalarMultiply(a: double) : Point + scalarDivide(a: double) : Point + dotProduct(point : Point) : double + linearInterp(point : Point, scale : double) : Point + magnitude( ) : double + normalise( ) : Point + crossProduct(point : Point) : Point + equals(point : Point) : boolean

Figure 12: Point Class

This class designed to model 3D points with x, y and z coordinates. It contains three instance variables which are x(double), y(double) and z(double).

Point( ) is a default constructor that construct a point at the default location of (0,0,0). Point(point : Point) would be an overloaded constructor that construct a point with the given point. Point(x : double, y : double, z : double) is an overloaded constructor that construct a point with the given x, y and z coordinates.

getX( ) : double, getY( ) : double and getZ( ) : double is a getter for the instance variables x, y and z.

Negate, add, subtract, scalarMultiply, scalarDivide, dotProduct, linearInterp, magnitude, normalize, crossProduct and equals are the function that is used to do the mathematics between different points. Negate is to make all x, y and z coordinates in negative value; Add is to add two points together; Subtract is to minus between two points; Scalar multiply and scalar

divide are to scale between two points; Dot product is created to find the magnitude of a vector; Linear interpolation is to interpolate towards the direction of the point that is working in. It is used to reconstruct polygons that have been partially removed from the solid; magnitude is just to find the magnitude of a vector; Normalise is to find the normal of the magnitude; Cross product is to find a vector which is perpendicular to the plane spanned by two vectors; Equal is just an equal function. This function is a getter for troubleshooting purpose.

## 6.2. Rectangle3D Class

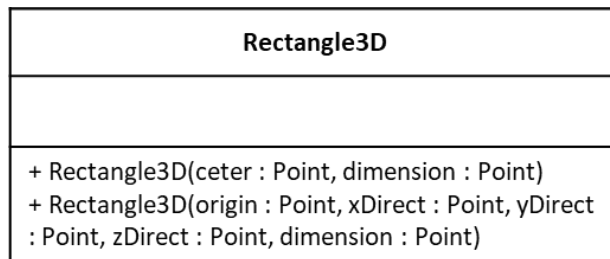


Figure 13: Rectangle3D Class

Rectangle3D is just inherited class from Solid. It is just there to construct a solid type of rectangle. All methods of Solid can be used on Rectangle3D.

Rectangle3D(center : Point, dimension : Point) is a constructor that construct a 3D rectangle with center point and dimensions of x, y and z. In this method, the array is four points for each rectangle surface of the 3D rectangle. There would be six surfaces in total. The normal is next to the position array indicate the surface direction. After assigning the position and normal, a loop is created with four-point vertices to make up a rectangle, using the center of the rectangle passed in, and the dimensions required. After the 4 vertices are obtained, a Polygon is created from these vertices.

To explain the loop in detail, for the first constructor:

```
(Math.min(1, i & 1) - 0.5)
(Math.min(1, i & 2) - 0.5)
(Math.min(1, i & 4) - 0.5)
```

Depending on where the point is on the cube, the coordinate of the point is different relative to the center of the cube.

For (Math.min(1, i & 1) - 0.5), this statement multiplies either 0.5 or -0.5 to the dimension in the x direction when constructing a point on the cube. 'i' is the position of the point on the cube. It is labeled in the array above the loop. For example, position 0:

- (Math.min(1, i & 1) - 0.5) will give us -0.5
- (Math.min(1, i & 2) - 0.5) will give us -0.5

- $(\text{Math.min}(1, i \& 4) - 0.5)$  will give us  $-0.5$

This result in point 0 is at  $(\text{center} - 0.5 * \text{dimension.x}, \text{center} - 0.5 * \text{dimension.y}, \text{center} - 0.5 * \text{dimension.z})$ .

While for position 5:

- $(\text{Math.min}(1, i \& 1) - 0.5)$  will give us  $0.5$ ,
- $(\text{Math.min}(1, i \& 2) - 0.5)$  will give us  $-0.5$ ,
- $(\text{Math.min}(1, i \& 4) - 0.5)$  will give us  $0.5$ .

This give point 5 at  $(\text{center} + 0.5 * \text{dimension.x}, \text{center} - 0.5 * \text{dimension.y}, \text{center} + 0.5 * \text{dimension.z})$ .

`Rectangle3D(origin : Point, xDirect : Point, yDirect : Point, zDirect : Point, dimension : Point)` is to construct 3D rectangle with origin point, directions of x, y and z as well as dimensions of x, y and z. The origin is the starting point of the construction. It must be placed at the bottom left back. A diagram is shown below for an example of where the origin point is set.

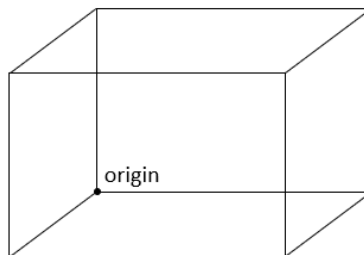


Figure 14: Rectangle3D Constructor

For the `xDirect`, `yDirect` and `zDirect` would be the x direction, y direction and z direction from the origin respectively. For the dimension, it only allows positive dimension for this construction.

To explain detail how the allocation of each vertex in the constructor, a diagram is shown below to have the idea of how it will look like.

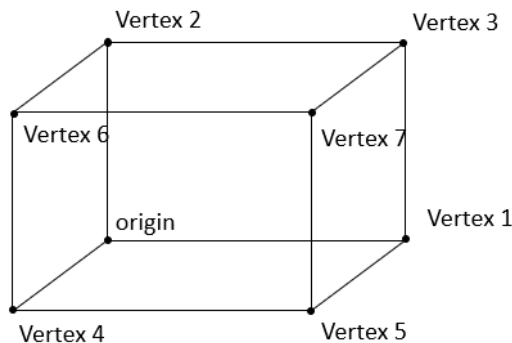


Figure 15: Rectangle3D Allocation

The array list has total of six surfaces. It will categorise which four points are to form a surface.

## 6.3. Cylinder Class

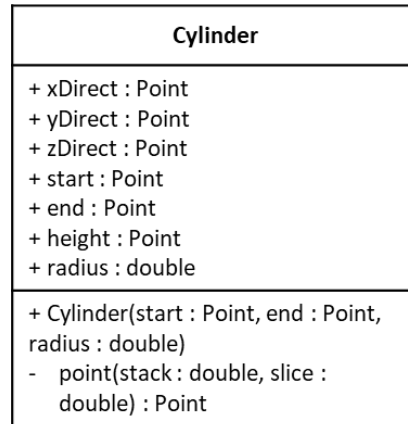


Figure 16: Cylinder Class

Cylinder is also an inheritance class from Solid. Same as the Rectangle3D class, it is also to construct a solid type of cylinder. All methods of Solid can be used on Cylinder. In this cylinder class, it models a cylinder with x direction, y direction and z direction point, start and end point, height as well as radius.

Cylinder(start : Point, end : Point, radius : double) is to construct a cylinder with start point, end point and radius. The slices are arbitrary. The higher the number of slices, the smoother the cylinder. However, after numerous of testing, I have set 24 as the number of slices. Height is also to normalise when it is assigned to zDirection. It is easier to handle with the further calculations below using the zDirection as calculations with normalised vectors.

A Boolean is also created to check if the y component in z is more than 0.5. If it is, the xDirection is set perpendicular to both zDirection and (1,0,0), else, xDirection would be perpendicular to both zDirection and (0,1,0).

A normalisation is also made for easy calculation later. These are just arbitrary directions and does not contribute to the magnitude of the cylinder. As for the crossProduct, it is to get the vector perpendicular to zDirection and the vector above. It is also explained at the previous paragraph.

A loop is created. As each slice, there is a top surface, a bottom surface, and a side surface. The top and bottom surfaces are triangles. and the side surface is a rectangle. Hence, three polygons are added for three slices. t0 and t1 are the start and end point of the slice radially.

Point(stack : double, slice : double) is to returns the absolute position of a point on the cylinder that is used to construct the polygons. Stack input indicates if this point is on the bottom or the top of the cylinder. Slice input is referring to t0 or t1, which is the boundaries of the slice. 'out' is the radial magnitude from the center of the cylinder and 'pos' is the position.

## 6.4. Polygon Class

Polygon
# vertices : List<Polygon> - normal : Point - w : double - epsilon : double (readOnly)
+ Polygon(vertices : List<Point>) + Polygon(polygon : Polygon) + flip( ) : void + getVertices( ) : List<Point> + getNormal( ) : Point + getW( ) : double + splitPolygon(polygon : Polygon, coplanarFront : List<Polygon>, coplanarBack : List<Polygon>, front : List<Polygon>, back : List<Polygon>) : void

Figure 17: Polygon Class

This class stores a polygon object that is made up of points on a single plane. Two constructors are available for this class.

Epsilon is a constant. It is a threshold to separate coplanar from back and front surfaces.

Polygon(vertices : List<Point>) takes a set of points or vertices, calculates the normal of the polygon, and stores the vertices in a list.

Polygon(polygon : Polygon) returns a new object that has a deep copy of every item in the input polygon, including the vertices in the list.

flip( ) : void is used to flip this polygon. Hence, if before it is an outside facing polygon with normal (-1,0,0), flip will make it face inside with a normal (1,0,0). The points are still in the same location. They are just rearranged to maintain the convention of normal calculation with the vertices.

getVertices( ) : List<Point>, getNormal( ) : Point, and getW( ) : double are just getters.

splitPolygon(polygon : Polygon, coplanarFront : List<Polygon>, coplanarBack : List<Polygon>, front : List<Polygon>, back : List<Polygon>) : void. In this categorizes the input polygon, whether it is in front or at the back of this polygon where the object it is operated on, it will add the input polygon to the appropriate list. The method is used in conjunction with clipPolygons of the Node class.

## 6.5. Node Class

<b>Node</b>
- polygon : Polygon - front : Node - back : Node - polygons : List<Polygon>
+ Node(polygons : List<Polygon>) + Node(node : Node) + Node( ) + getPolygon( ) : Polygon + getFront( ) : Node + getBack( ) : Node + getPolygons( ) : List<Polygon> + Invert( ) : void + clipPolygons(polygons : List<Polygon>) : List<Polygon> + clipTo(bsp : Node) : void + allPolygons( ) : List<Polygon> + build(polygons : List<Polygon>) : void

Figure 18: Node Class

This class is storing the node of the tree constructor. Each node object is categorized as either a back node or a front node. All the node objects in a Solid will be compared to its first Polygon (firstPolygon) in the list. A back node object contains polygons that are at the back of the firstPolygon. A front node object contains polygons that are in front of the firstPolygon. A node object may contain 0 or more polygons. If there are more than 1, it indicates that all the polygons in that particular node are of the same plane. This categorization uses the function from the Polygon class which are splitPolygons.

Node(polygons : List<Polygon>) is to create a list for all polygons. Node(node : Node) is to check all the front and back node for all polygons and store in a list. This is explain at the previous paragraph. Node() is just an empty constructor for composited class to use.

getPolygon( ) : Polygon, getFront( ) : Node, getBack( ) : Node and getPolygons( ) : List<Polygon> are just getters.

invert( ) : void is a function that flip the polygons in the node, and makes it a front node if it was a back node and it is vice versa. It is to make a hole into a solid space, and a solid space into a hole.

clipPolygons(List<Polygon>) in this method, it removes all polygons in the parameter List<Polygon> that are inside this object. This is done by categorising the polygons in the list into back or front polygons. Then, it is called recursively with the following below:

```
front = this.front.clipPolygons(front);  
back = this.back.clipPolygons(back);
```

The full process is as follows. First, use first Node of this solid. Then it will categorize all polygons in list input into front or back polygons when compared to the polygon in this Node. After that, the 'call front = this.front.clipPolygons(front)' is called uses a Node with polygons that are in front of the polygons in this Node. It checks whether the input polygons categorized as front, is also in front of this Node. This is checked recursively. If a polygon is in front of all Nodes in this solid, this polygon is kept. While for "call back = this.back.clipPolygons(back)", this method call is the same as the call front method, but it checks for polygons that are back. This is also recursively check and it only keeps a polygon if it is at the back of all Nodes in this solid. Finally, all the remaining polygons are appended into a list and returned.

This process removes all polygons that are in between the polygons of this solid. Polygons that are in front of some Nodes, but at the back of other Nodes indicate that it is inside the solid. Therefore, it is removed.

clipTo(Node bsp) removes all polygons in this object that is inside the input Solid represented by the Node. It uses helper function clipPolygons.

allPolygons() will returns a list of all the polygons in this chain of Nodes.

build(List<Polygons>) builds the tree of the Solid with the list of its polygons. If a polygon is in front of a reference polygon, it is moved to the front Node. If a polygon at the back of the reference polygon, it is moved to the back Node. The reference polygon is the first polygon in the list of polygons of that Node.



## 6.6. Solid Class

Solid
# polygons : List<Polygon>
+ Solid(polygons : List<Polygon> + Solid(solid : Solid) + Solid( ) + getPolygons( ) : List<Polygon> + union(solid : Solid) : Solid + subtract(solid : Solid) : Solid + intersect(solid : Solid) : Solid +toTriangles( ) : List<Polygon>

Figure 19: Solid Class

This class stores a solid, which is a list of polygons. It has three constructors.

Solid(polygons: List<Polygon>) stores the polygons in this object. Solid(solid : Solid) creates new object instances of everything in the input solid, including the points in the polygon list. Solid( ) is just an empty constructor for inherited classes to use.

The union, subtract, and intersect are the three functions that help create the new solids requirement. Union adds two solids together, where all surfaces or polygons that intersect between the two solids are removed; Subtract removes a solid from another; Intersect keeps only surfaces that are inside each other. Figure below show two sample shapes use to describe the formation of union, subtract, and intersect function.

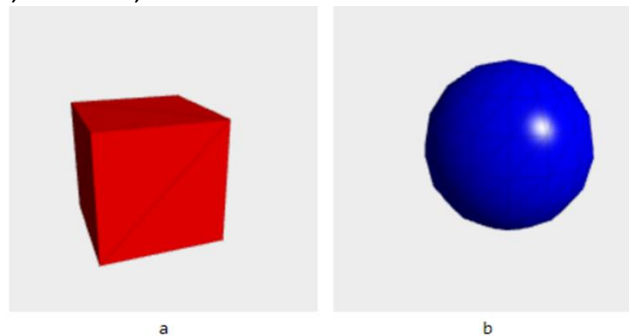


Figure 20: Shape Demonstration [16]

To further describe how the union function work, firstly, I create two trees with nodes having a set of coplanar surfaces. Then, it removes all of a's surfaces that are in b and it also removes all of b's surfaces that are in a. To keep only 1 set of coplanar surfaces from both solids, coplanar surfaces in b will be removed. To do this, b is inverted, then it removes all inverted b's surfaces that are in a. b is then inverted again and a new solid is built by combining the remaining surfaces from both solids. [16]

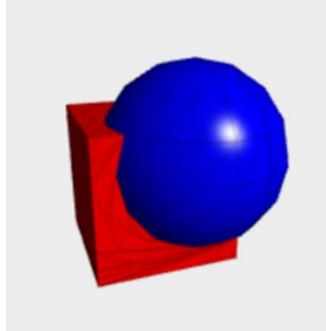


Figure 21: Union Combination Solid Shape [16]

Similar concept is applied to subtract function as well. I create a tree for each solid with nodes having a set of coplanar surfaces. Then, it removes all outer surfaces of a that is inside b and it also removes all surfaces of b that is inside a. The duplicate coplanar surfaces will be removed. A new solid is built using all remaining surfaces from both solids. [16]

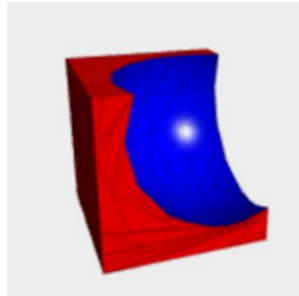


Figure 22: Subtract Combination Solid Shape [16]

As for the intersect function, I also create a tree for each solid with nodes having a set of coplanar surfaces. Instead of removing the surface from one another, first invert a, then, remove all of b's surfaces that are in a. After that, b is inverted and remove all of a's surfaces that are in b. Then, all of b's surfaces also need to remove from a. Then a is inverted to form a new solid using all the remaining surfaces from both solid. [16]

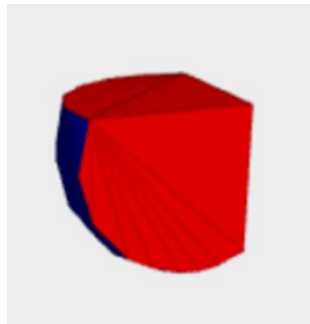


Figure 23: Intersect Combination Solid Shape [16]

toTriangles(List<Polygons>) is to convert any non-triangle polygons in the input list to triangles. The arrangement of vertices in triangle maintains normal direction.

The inner workings of these functions require the use of the Node class which will be explained in the previous section.

## 6.7. StlWriterUtil Class

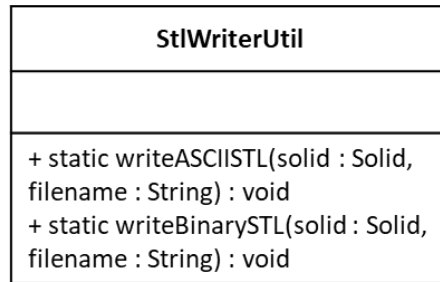


Figure 24: StlWriterUtil Class

StlWriterUtil class is a class to convert solid to the ascii or binary format.

For the ascii format is it build in this following [3]:

```
solid name
facet normal ni nj nk
  outer loop
    vertex v1x v1y v1z
    vertex v2x v2y v2z
    vertex v3x v3y v3z
  endloop
endfacet
```

writeASCII STL(solid : Solid, filename : String), this method will pass ASCII code of the solid and the designed file name and generate an STL file to view the final 3D diagram. An ASCII STL file will begin with a string name to name the solid. Then the file will continue to get all the information of the triangle with the format showed above. In a for loop, the facet normal is first store by getting all the normal from x, y, and z. Then we will get all the vertices with x, y and z as well. This will then wrap up with ending the loop and facet.

writeBinary STL(solid : Solid, filename : String) is a method to pass the binary code of the solid with the designed file name and generate an STL file to view the final 3D diagram. Unlike ASCII constructor, binary constructor does not start with solid because it may lead to the software to mistook it as an ASCII generator. Hence, the header would be a 4-byte little endian unsigned integer to indicate the number of triangles in the file. Then, a for loop is created to run all the triangle normal and vertices. After all the last triangle information have taken, it will end with flushing all the unuse bytes and close. The following below is a general idea of how binary STL file format look like. [3]

UINT8[80]	– Header	- 80 bytes
UINT32	– Number of triangles	- 4 bytes
foreach triangle		- 50 bytes:
REAL32[3]	– Normal vector	- 12 bytes
REAL32[3]	– Vertex 1	- 12 bytes
REAL32[3]	– Vertex 2	- 12 bytes
REAL32[3]	– Vertex 3	- 12 bytes
UINT16	– Attribute byte count	- 2 bytes
end		

## 7. STL Model

In this session, I am going to show you some of the STL models that I have generated from the library that I have created.

### 7.1. Calcium Lactate with Sodium Alginate



Figure 25: Calcium Lactate with Sodium Alginate Interior

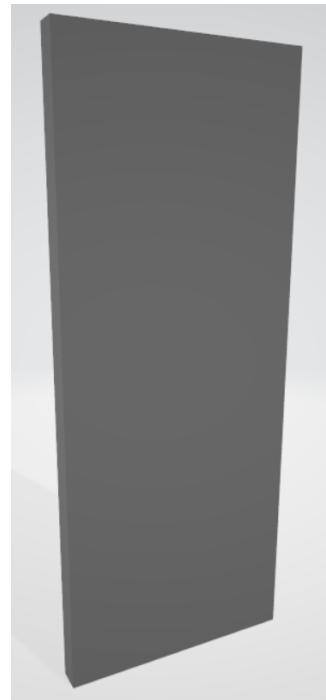


Figure 26: Calcium Lactate with Sodium Alginate Exterior

Above show the Calcium Lactate with Sodium Alginate design and how it formed the 3D image. Figure 25 is the interior look of the chip while figure 26 is the exterior look of the chip. We would need to design both interior and exterior chips separately. After the implementation has done, we would use a function in solid class which is subtract function to combine both figures above. To form the Calcium Lactate with Sodium Alginate chip, simply subtract the interior from the exterior to form channel in the chip. Below is the sample look and detail explanation of how the STL model is formed.

#### **Without Base**

This is one of the STL model that I have created. All coordinates are guesses. It might not be accurate to scale of original drawings.

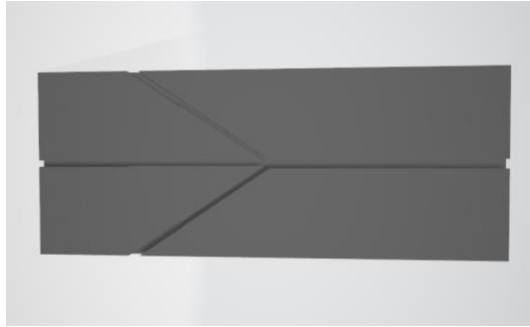


Figure 27: Calcium Lactate with Sodium Alginate Top View - Baseless

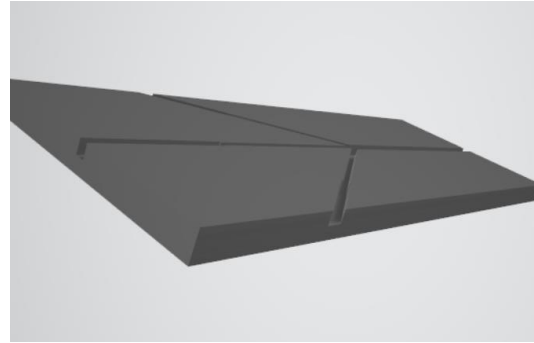


Figure 28: Calcium Lactate with Sodium Alginate Side View 1 - Baseless

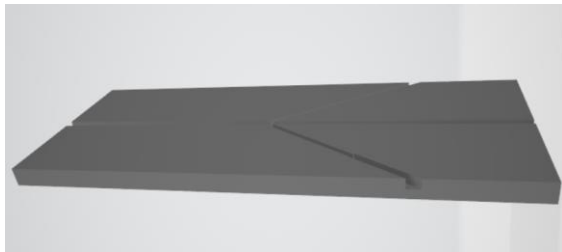


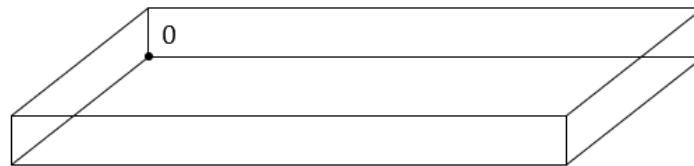
Figure 29: Calcium Lactate with Sodium Alginate Side View 2 - Baseless



Figure 30: Calcium Lactate with Sodium Alginate Side View 3 - Baseless

The step by step of the construction is as follow:

1. Construct the large solid rectangle.
  - I started from origin, (0, 0.05, 0), while for the arbitrary point can be anything.
  - Then, set x, y and z direction from the origin.



- When constructing, the origin must be at position 0.
  - x, y, and z direction are arbitrary, depending on the coordinate system that use.
  - In my design, I use x as right from origin, z as up from origin, and y as front from origin.
2. Construct the top channel.
    - For this origin is at the middle of the first solid, offset to the left by half of channel width. The dimensions are estimated.
  3. Construct bottom wider channel.
    - For this, the origin is at the same x, and y coordinates as solid 2, z coordinate is at the most bottom. The dimensions are also estimated.
  4. Construct bottom narrower channel.

- For this, the origin is at the same x and y coordinates as solid 2, z coordinate is set at the end of solid 3. The z coordinate is set at 2 because I ended solid 3 at 2.
5. Construct diagonal channel.
    - For this, origin is set at left most edge of solid. z coordinate is a guess. I set at 2.
    - For x direction, set as the diagonal towards the middle of solid 1.
    - For z direction, calculate the normal with cross product between x and y direction vectors. All the dimensions are estimated.
  6. Construct narrower diagonal channel.
    - For this, origin is set at the left most edge of the solid 1. z coordinate is slightly higher than solid 5.
    - For x direction, set as the diagonal towards the middle of solid 1.
    - For z direction, calculate the normal with cross product between x and y direction vectors. All the dimensions are estimated.
  7. Construct the other diagonal channel.
    - For this, origin is set at the right most edge of solid 1. z is again a guess. I set at 2.
    - For this solid, the z Direction is the vector to the middle of solid 1.
    - For x direction, it is calculated using cross product of y and z direction vectors. All the dimensions are estimated.
  8. Construct the other narrower diagonal channel.
    - For this, the origin is set at the right most edge of solid 1. Z is set slightly higher than solid 7.
    - For this solid, the z direction is the vector to the middle of solid 1.
    - For x direction, it is calculated using cross product of y and z direction vectors. All the dimensions are estimated.
  9. Compound all the construction to generate 3D diagram.
    - new solid is the name of the generation. It is the combination of solids 2 to 8 that are subtracted from solid 1.
  10. List of triangles are obtained from the list of polygons in new solid.
  11. Pass triangles to write stl util.

## With Base

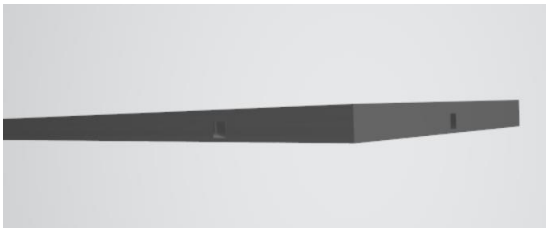
This is the same concept as the previous Calcium Lactate with Sodium Alginate Baseless design. The different is the input value for the construction of large solid rectangle. Instead of using the origin as (0, 0.5, 0) now I use (0, 0, 0) as the origin to achieve the Calcium Lactate with Sodium Alginate with base.



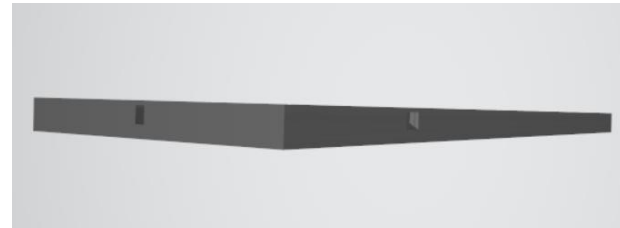
*Figure 31: Calcium Lactate with Sodium Alginate Top View with Base*



*Figure 32: Calcium Lactate with Sodium Alginate Side View 1 with Base*



*Figure 33: Calcium Lactate with Sodium Alginate Side View 2 with Base*



*Figure 34: Calcium Lactate with Sodium Alginate Side View 3 with Base*

## 7.2. Ring Normal

This is another STL model that I have created. It is almost the same process as Calcium Lactate with Sodium Alginate design. All coordinates are guesses. It might not be accurate to scale of original drawings. Same as the design for Calcium Lactate with Sodium Alginate chip, I also design the interior and exterior of the ring normal separately.



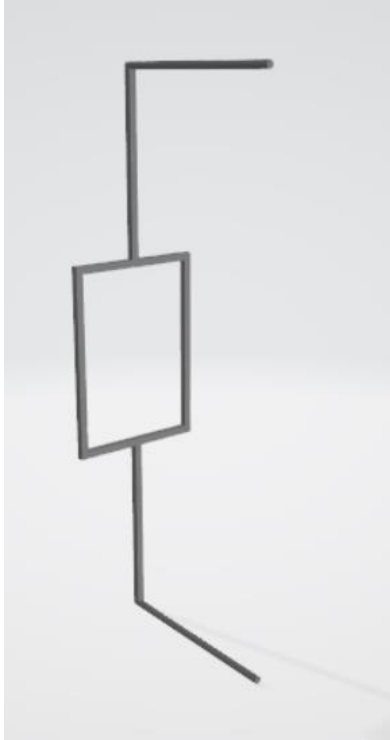


Figure 35: Ring Normal Interior



Figure 36: Ring Normal Exterior

To form the Ring Normal chip, I also subtracted the ring normal interior to ring normal exterior to form the chip. Below shows the end result of how ring normal look like and detail explanation is also done to explain how it created.

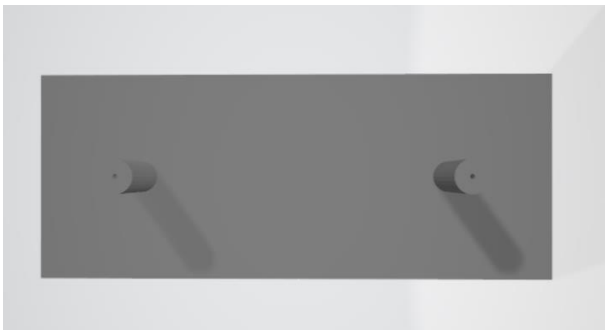


Figure 37: Ring Normal Top View

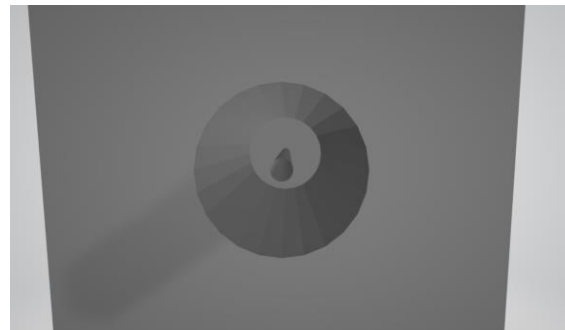


Figure 38: Ring Normal Cylinder View

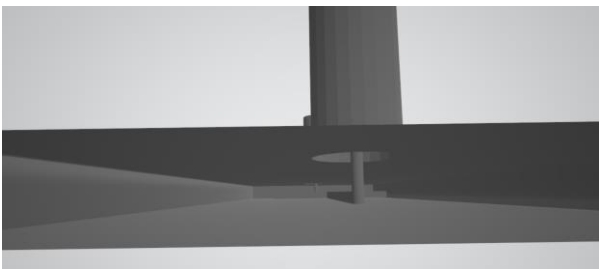


Figure 39: Ring Normal Interior View 1

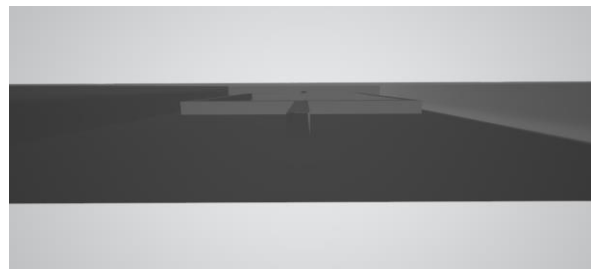


Figure 40: Ring Normal Interior View 2

The step by step of the construction is as follow:

1. Construct the large solid rectangle.
  - This is the same as the above design, I started from origin,  $(0, 0, 0)$ , while for the arbitrary point can be anything.
  - Then, set x, y and z direction from the origin.



- When constructing, the origin must be at position 0.
  - x, y, and z direction are arbitrary, depending on the coordinate system that use.
  - In my design, I use x as right from origin, z as up from origin, and y as front from origin.
2. Construct the first cylinder.
    - For this, I assign the start point as  $(2, 0.2, 2)$  and the end point is 2 unit different which would be at the point of  $(2, 2.2, 2)$ . The radius is estimated, and all the dimensions are also estimated.
  3. Construct the second cylinder.
    - This is the same as the first cylinder. But this time, I change the position of start point to  $(2, 0.2, 8)$  and the end point to  $(2, 2.2, 8)$ . The radius is estimated, and all the dimensions are also estimated.
  4. Construct the first hole.
    - This is the same as the cylinder design above. The reason of naming it as hole is because I am constructing the inlet and outlet hole in the cylinder. Hence the dimension would be smaller than the cylinder. All the dimensions are estimated.
  5. Construct the second hole.
    - This is also the same as the cylinder and hole designs above. The different is the position of the hole. The dimension would be smaller than the cylinder. All the dimensions are estimated.
  6. Construct the hole connectors.
    - This is to construct a connector between the inlet outlet and the loop at the middle. For this, the origin is at the same x and y coordinates for the first and second holes, z coordinate is set 1 unit different with respect to the inlet outlet hole. All the dimensions are estimated.

7. Construct the loop.
  - For this, the top loop and bottom loop have the same x and y coordinates while the left loop and right loop have the same y and z coordinates. The z coordinate is different to locate the top and bottom loop and the x coordinate is different to locate the left and right loop. All the dimensions are estimated.
8. Compound all the construction to generate 3D diagram.
  - new solid is the name of the generation. It is the combination of solids 4 to 7 that are subtracted from union of solids 1 to 3.
9. List of triangles are obtained from the list of polygons in new solid.
10. Pass triangles to write stl util.

### 7.3. Mixer

This is another STL model that I have created to show some different shape that can be generated from the library itself. The idea of building it is the same as what I explain it at the previous model.

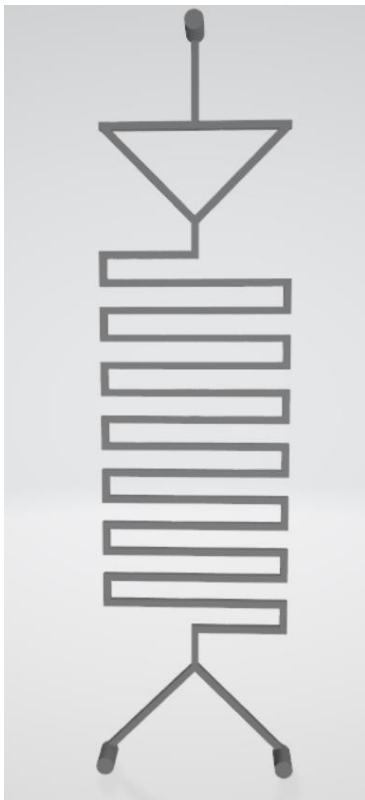


Figure 41: Mixer Interior

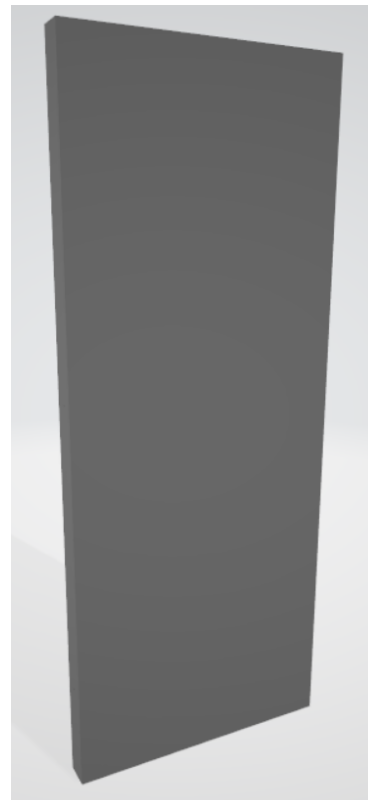


Figure 42: Mixer Exterior

## Without Base

Same as what I did earlier, I also design the mixer to show the different between how with and without base look like. The detailed explanation is also showed below.

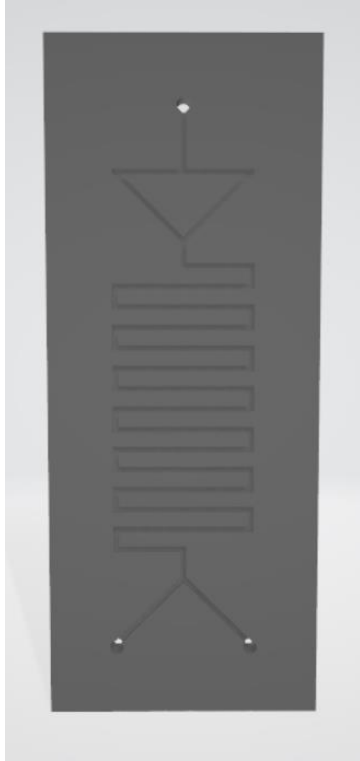


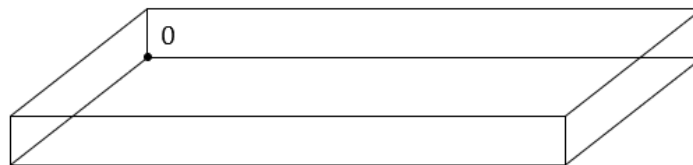
Figure 43: Mixer Bottom View - Baseless



Figure 44: Mixer Top View - Baseless

The step by step of the construction is as follow:

1. Construct the large solid rectangle.
  - This is the same as the above design, I started from origin,  $(0, 0.05, 0)$ , while for the arbitrary point can be anything.
  - Then, set x, y and z direction from the origin.



- When constructing, the origin must be at position 0.
- x, y, and z direction are arbitrary, depending on the coordinate system that use.
- In my design, I use x as right from origin, z as up from origin, and y as front from origin.

2. Construct the first hole.
  - For this, I assign the start point as (1, 0.05, 1) and the end point is 0.45 unit different which would be at the point of (1, 0.5, 1). The radius is estimated, and all the dimensions are also estimated.
3. Construct the second hole.
  - This is the same as the first hole. But this time, I change the position of start point to (3, 0.05, 1) and the end point to (3, 0.5, 1). The radius is estimated, and all the dimensions are also estimated.
4. Construct the third hole.
  - This is the same as the first and second holes. But this time, I change the position of start point to (2, 0.05, 9) and the end point to (2, 0.5, 9). The radius is estimated, and all the dimensions are also estimated.
5. Construct diagonal channel.
  - For this, origin is set at left most edge of solid. z coordinate is a guess. I set at 1.
  - For x direction, set as the diagonal towards the middle of solid 1.
  - For z direction, calculate the normal with cross product between x and y direction vectors. All the dimensions are estimated.
6. Construct the other diagonal channel.
  - For this, origin is set at the right most edge of solid 1. z is again a guess. I set at 1.
  - For this solid, the z Direction is the vector to the middle of solid 1.
  - For x direction, it is calculated using cross product of y and z direction vectors. All the dimensions are estimated.
7. Construct the diagonal connectors.
  - This is to construct a connector between the two inlets and the s channel at the middle. For this, the center set as the middle of both inlets which is at (2, 0.09, 2.25). The dimension of the connector would be the same as the diagonal connector the only different is the z direction of the length. All the dimensions are estimated.
8. Construct the hole connectors.
  - This is to construct a connector for the outlet and the triangular loop. For this, the center is set same as the outlet position which is at (2, 0.09, 8.5). The dimension of the connector would be the same as the rest of the channels, only different is the z direction of the length. All the dimensions are estimated.
9. Construct the s channel.
  - This is to construct the s channel for the mixer. All the horizontal channels would be the same dimension but only different is the position in z direction. While for the vertical, all the vertical channels are also same dimension but the only different is the left

vertical would be at 1 in x direction and 3 in x direction and the position in z direction is also different to connect all the horizontal channel together to form the s channel.

10. Construct the triangular loop.

- This is made up with 2 diagonal channel and one horizontal channel. The concept is the same as what I did for the diagonal channel earlier but only changing the position and the length of the channels.

11. Compound all the construction to generate 3D diagram.

- new solid is the name of the generation. It is the combination of solids 4 to 7 that are subtracted from union of solids 1 to 3.

12. List of triangles are obtained from the list of polygons in new solid.

13. Pass triangles to write stl util.

## With Base

Below shows the mixer model with base. Everything for the construction would be the same except for the size of the chip. Instead of using (0, 0.05, 0) as the origin, now I would use (0, 0, 0) as the origin.



Figure 45: Mixer Top View with Base

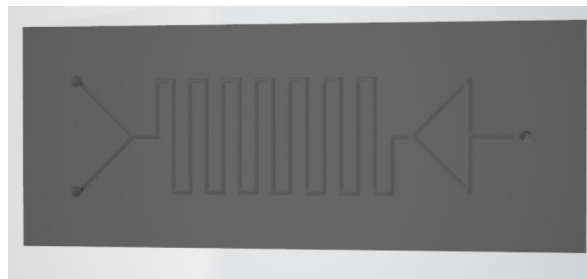


Figure 46: Mixer Bottom View with Base



Figure 47: Mixer interior View 1 with Base

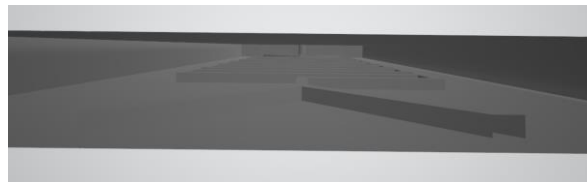


Figure 48: Mixer interior View 2 with Base

## 8. Conclusion

In conclusion, designing a library is definitely not an easy task. It takes a lot of effort on researching before we can start on the actual implementation. It is especially difficult for me as Java is a new programming language that I have not learn before. Other than this, learning the object-oriented programming is also another challenge because algorithm needs to be understood to start on the implementation.

As for the development of the coding, there is a lot of try and error to test each and every classes. After numerous try and a library is finally created. From this thesis, I have learnt that self-learning is very important as many things we need to figure it out ourselves. I also pick up a new skill which is Java programing language.

There are still some things to improve for this library, as due to the time constraint, I am unable to create the feature of correcting the values or algorithms of the 3D images after generating the STL file. Writers must manually change from the java code itself and run again the program. Hence, my suggestion for the improvement of this project would be adding a feature for amending the 3D imaging.

## 9. References

- [1] elveflow, "Elveflow," [Online]. Available: <https://www.elveflow.com/microfluidic-reviews/general-microfluidics/microfluidics-and-microfluidic-device-a-review/#:~:text=The%20microfluidic%20technology%20has%20found,characteristic%20size%20as%20biological%20cells>. [Accessed 11 5 2021].
- [2] T. F. E. Wikipedia, "Wikipedia, The Free Encyclopedia," [Online]. Available: [https://en.wikipedia.org/wiki/Library\\_\(computing\)#:~:text=In%20computer%20science%2C%20a%20library,classes%2C%20values%20or%20type%20specifications](https://en.wikipedia.org/wiki/Library_(computing)#:~:text=In%20computer%20science%2C%20a%20library,classes%2C%20values%20or%20type%20specifications). [Accessed 11 5 2021].
- [3] T. F. E. Wikipedia, "Wikipedia, The Free Encyclopedia," [Online]. Available: [https://en.wikipedia.org/wiki/Java\\_Class\\_Library](https://en.wikipedia.org/wiki/Java_Class_Library). [Accessed 11 5 2021].
- [4] T. F. E. Wikipedia, "Wikipedia, The Free Encyclopedia," [Online]. Available: [https://en.wikipedia.org/wiki/STL\\_\(file\\_format\)](https://en.wikipedia.org/wiki/STL_(file_format)). [Accessed 11 5 2021].
- [5] C. Bipat, "Newyork Engineers," 7 2 2021. [Online]. Available: <https://www.ny-engineers.com/blog/types-of-tiles-used-in-flooring>.
- [6] D. Chakravorty, "Craftcloud," 14 2 2019. [Online]. Available: <https://all3dp.com/what-is-stl-file-format-extension-3d-printing/>.
- [7] P. Bourke, 10 1999. [Online]. Available: <http://paulbourke.net/dataformats/stl/>.
- [8] W. Commons, "Wkimedia Commons," [Online]. Available: <https://commons.wikimedia.org/wiki/File:Octants.png>. [Accessed 11 5 2021].
- [9] W.H.Lipscomb, "ResearchGata," 8 2005. [Online]. Available: [https://www.researchgate.net/figure/Generating-geodesic-grids-by-recursive-bisection-and-projection\\_fig7\\_307561876](https://www.researchgate.net/figure/Generating-geodesic-grids-by-recursive-bisection-and-projection_fig7_307561876).
- [10] A. Chen, "C-MAC Industries Pty Ltd," 18 3 2019. [Online]. Available: <https://www.cmac.com.au/blog/5-vital-things-about-stl-file-format-3d-printing#:~:text=Binary%20or%20ASCII,to%20use%20for%203D%20printing>.
- [11] Guru99, "Guru99," [Online]. Available: <https://www.guru99.com/java-platform.html#1>. [Accessed 11 5 2021].
- [12] A. Altvater, "Stackify," 5 4 2017. [Online]. Available: <https://stackify.com/oops-concepts-in-java/#:~:text=OOP%20concepts%20in%20Java%20are,encapsulation%2C%20inheritance%2C%20and%20polymorphism.&text=Basically%2C%20Java%20OOP%20concepts%20let,of%20them%20without%20compromising%20security>.



- [13] Fluigent, "Fluigent," [Online]. Available: <https://www.fluigent.com/resources/microfluidic-expertise/what-is-microfluidic/microfluidic-chip-history/#:~:text=Microfluidic%20chips%20are%20devices%20used,from%20one%20place%20to%20another>. [Accessed 11 5 2021].
- [14] Fluigent, "Fluigent," [Online]. Available: <https://www.fluigent.com/resources/microfluidic-expertise/what-is-microfluidic/how-to-choose-a-microfluidic-chip/>. [Accessed 11 5 2021].
- [15] R. D. M. C. E. N. C. F. C. Y. C. S. L. N. B. A. P. J. F. R. F. S. J. I. D. E. Novak, "Jove," 20 10 2018. [Online]. Available: <https://www.jove.com/t/58151/scalable-fabrication-stretchable-dual-channel-microfluidic-organ>.
- [16] github, "github," [Online]. Available: <https://evanw.github.io/csg.js/>. [Accessed 11 5 2021].