Resolution-Aware Slicing of CAD Data for 3D Printing

ISIDORE ONYEAKO

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements for the degree

Master of Applied Science in Biomedical Engineering

Ottawa-Carleton Institute for Biomedical Engineering

School of Electrical Engineering and Computer Science

University of Ottawa

Ottawa, Ontario, Canada

Abstract

3D printing applications have achieved increased success as an additive manufacturing (AM) process. Micro-structure of mechanical/biological materials present design challenges owing to the resolution of 3D printers and material properties/composition. Biological materials are complex in structure and composition. Efforts have been made by 3D printer manufacturers to provide materials with varying physical, mechanical and chemical properties, to handle simple to complex applications. As 3D printing is finding more medical applications, we expect future uses in areas such as hip replacement - where smoothness of the femoral head is important to reduce friction that can cause a lot of pain to a patient. The issue of print resolution plays a vital role due to staircase effect. In some practical applications where 3D printing is intended to produce replacement parts with joints with movable parts, low resolution printing results in fused joints when the joint clearance is intended to be very small. Various 3D printers are capable of print resolutions of up to 600dpi (dots per inch) as quoted in their datasheets. Although the above quoted level of detail can satisfy the micro-structure needs of a large set of biological/mechanical models under investigation, it is important to include the ability of a 3D slicing application to check that the printer can properly produce the feature with the smallest detail in a model. A way to perform this check would be the physical measurement of printed parts and comparison to expected results. Our work includes a method for using ray casting to detect features in the 3D CAD models whose sizes are below the minimum allowed by the printer resolution. The resolution validation method is tested using a few simple and complex 3D models. Our proposed method serves two purposes: (a) to assist CAD model designers in developing models whose printability is assured. This is achieved by warning or preventing the designer when they are about to perform shape operations that will lead to regions/features with sizes lower than that of the printer resolution; (b) to validate slicing outputs before generation of G-Codes to identify regions/features with sizes lower than the printer resolution.

Acknowledgement

I acknowledge the valued coaching, encouragement and support provided by my Supervisor, Prof. Wonsook Lee; by encouraging creative thinking and non-conventional methods for dealing with the research projects involved in this program.

Table of Contents

Abstract ii
Acknowledgement iii
Table of Contents iv
List of Figures vi
List of Tables viii
Chapter 1. Introduction 1
1.1 Motivation 2
1.2 Overview of Our System 3
1.3 Our Contributions 4
1.3.1 Resolution Validation Method 4
1.3.2 SlicingResValidator 5
1.3.3 Android STL 3D CAD Viewer and STL Parser 5
1.4 Thesis Organization 5
Chapter 2. Literature Review 6
2.1 Medical Applications of 3D printing 6
Titanium, 3D-printed prosthetic jaw implanted in Melbourne man in Australian first surgery[67] 8
2.2 Basic 3D Printing Process 8
2.3 3D Printing Methods 9
2.3.1 Fused Filament Fabrication (FFF)/Fused Deposition Model (FDM) 9
2.3.2 Electron Beam Freeform Fabrication (EBFF)/Electron Beam Melting 11
2.3.3 Stereolithography (SLA) 11
2.3.4 Selective Laser Sintering (SLS) 12
2.3.5 Direct Metal Laser Sintering (DMLS) 13
2.4 3D Printing CAD Data 15
2.5 3D Printing Slicing and G-Code 18

2.5.1 Slicing 18
2.5.2 G-Code Generation 21
2.6 Slicing and 3D Printing Quality Assurance 23
2.7 Resolution-Aware Slicing 25
2.7.1 Printer Resolution 25
2.7.2 Intersection of two lines and calculation of feature sizes 29
Chapter 3. Methodology and system 33
3.1 Objectives 33
3.2 Data structure for Managing Slicing Data 33
3.3 Ray Casting of 3D slice data 36
3.3.1 SVG Slice input sets 36
3.3.2 Pseudo-code/algorithm for line intersection 38
3.3.3 SlicingResValidator, User Interface and Workflow 39
Chapter 4. Results and Validation 42
4.1 Handling of units of measurement 42
4.2 Validation 43
4.3 Results 44
4.4 Discussion 61
Chapter 5. Conclusion 64
References 66
Appendix A: User Interface 72
Appendix B: Source Code 73
Abbreviations 88

List of Figures

Figure 1-1: Overview of the system..... 4
Figure 2-1: Rapid Prototyping model in a hospital (adapted from [53]) 6
Figure 2-2: Few examples of recent medical applications of 3D printing..... 8
Figure 2-3: 3D Basic Printing Workflow [14]..... 9
Figure 2-4: Fused Filament Fabrication (FFF)/Fused Deposition Model (FDM) Process [16]10
Figure 2-5: Fused Filament Fabrication (FFF)/Fused Deposition Model (FDM) Process 2 [17][19]..... 10
Figure 2-6: Schematic of Electron Beam Freeform Fabrication/Electron Beam melting (adapted from [19]) 11
Figure 2-7: Stereolithography (SLA) (adapted from [20]) 12
Figure 2-8: Selective Laser Sintering (SLS) [23] 12
Figure 2-9: Direct Metal Laser Sintering (DMLS) [9][26] 13
Figure 2-10: Standard Tessellation language (STL) File and CAD model [24] 16
Figure 2-11: ASCII Standard Tessellation language (STL) File format 16
Figure 2-12: Binary Standard Tessellation language (STL) File format 17
Figure 2-13: STL Facet Normal [31] 17
Figure 2-14: (a) Tessellated cube with 12 facets (b) a sliced cube [32]..... 18
Figure 2-15: Possible intersection cases [32]..... 19
Figure 2-16: (a): A model (blue) with one slice (red) (b): Slice and its contours..... 20
Figure 2-17: Slicing data structure 20
Figure 2-18: (a) Schematic illustration of G-Code generation and (b) resultant code of the algorithm [32]. 21
The code snippet in Figure 2-19(b) includes some popular G-Code commands. Below is a description of few G-Code commands: 22
Figure 2-20: Build stages on executing G-Code 22
Figure 2-21: Comparison of Minimum Feature Size (adapted from [22]) 26
Figure 2-22: Comparison of Minimum Layer Thickness (adapted from [22])..... 27
Figure 2-23: Effect of resolution on printability of features 28
Figure 2-24: Possible intersection of 2 lines. 30
Figure 2-25: Schematic of Ray casting to detect solid features for size comparison..... 31
Figure 3-1: Data Structure for slicing output 34

Figure 3-2: Algorithm for determining unprintable regions (defects). 35
Figure 3-3: Algorithm Browser rendering of SVG data for one slice of regular shapes cross sections. 37
Figure 3-4: SVG <g> output (top) for two circles (bottom) 38
Figure 3-5: SlicingResValidator Intersection Point Detector Algorithm 39
Figure 3-6 SlicingResValidator User Interface 41
Figure 4-1: 2D Grid Model for Validation 43
Figure 4-2: SlicingResValidator Output of Validation 2D Model check using x=3.0 and y-0.1 resolutions. The model did not detect any error as expected since the smallest feature is larger than 3.0 45
Figure 4-3: SlicingResValidator Output of Validation 2D Model check using x=4.0 and y-4.0 resolutions. Errors were detected in all the squares having sizes less than 4.0. Also errors were detected in gaps with sizes below the resolution values (blue). 45
Figure 4-4: SlicingResValidator Output of Validation 2D Model check using x=4.0 and y-2 resolutions. More Errors were detected in all the squares having sizes less than 4.0. As we reduced the resolution value in y direction by half (y=2), the algorithm detected more errors along the y direction by merely sampling (shooting) more rays. 46
Figure 4-5: 3D CAD Regular Geometrical shapes Checked with SlicingResValidator 48
Figure 4-6: Regression plot for SlicingResValidator outputs for a resolution of X=1 and y=1, 1.5, 2, 2.5... 50
Figure 4-7: Regression plot for SlicingResValidator outputs for a resolution of x=1, 1.5, 2, 2.5 51
Figure 4-8: 3D CAD Sample Tentacle End Support Checked with SlicingResValidator 51
Figure 4-9: Slice number 0 of Sample Tentacle End Support Checked with SlicingResValidator 52
Figure 4-10: Regression plot for SlicingResValidator outputs for Slice number 0 with a resolution of X=0.5 and y=1, 1.5, 2, 2.5... 53
Figure 4-11: 3D CAD Sample of Sinus Veins Checked with SlicingResValidator 54
Figure 4-12: Slice number 30 of Sample of Sinus Veins Checked with SlicingResValidator with resolution values x=1, y=0.5 55
Figure 4-13: Regression plot for SlicingResValidator outputs for Slice number 30 with a resolution of X=0.5 and y=1, 1.5, 2, 2.5... 56
Figure 4-14: Slice number 30 of Sample of Sinus Veins Checked with SlicingResValidator with resolution values x=1, y=0.5 57
Figure 4-15: Regression plot for SlicingResValidator outputs for Slice number 30 with a resolution of X=1, 1.5, 2, 2.5... and y=1 58
Figure 4-16: More 3D models evaluated by our application 61

List of Tables

Table 2-1: Comparison of Capabilities of 3D Printing [22] 14

Chapter 1. Introduction

With the dramatic evolution in medical imaging in the past few decades, radiological diagnosis has become less invasive. It is now possible to acquire high resolution three-dimensional image data in a short time. 3D visualization tools have been employed in several applications and they have put radiologists at the center of most clinical disciplines [5]. With the increasing processing power of computers and evolving imaging technologies, image guided surgeries has become common. This to a great extent gives an idea of what is possible with medical imaging. We are, however, limited by the use of flat screens for visualization of three dimensional imaging data. There are material and physical properties of an object that are better appreciated and understood with touch and feel. Three-Dimensional printing or rapid prototyping overcomes this limitation. A graspable object produced by 3D printing can be used to study various aspects of the problem being diagnosed [6].

The various 3D printing technologies fall under a few categories as an additive manufacturing process. Some technologies run materials in strands over an area guided by the CAD/CAM system at a given temperature for the material to fuse upon solidification. Fused deposition model (FDM)[7] is in this category. Another method heats powder materials to sinter (fuse) and subsequently wash away the excess. Selective laser sintering (SLS)[8] is in this category. Direct Metal Sintering (DMS)[9] heats metal at precise geometry to melt while SLA (Stereolithography) involves the hardening of liquid materials at precise geometry to form solids in contact with laser light[10].

What is common in all the 3D printing technologies is the use of 3D CAD data. This work focused on the quality of 3D printing process. In effect, the assumption is that the resolution of a 3D printer will affect the quality of the object it prints. This is evident in the fact that resolution has a direct relationship with the level of details. Our work introduced the need to apply the resolution of a printer as an additional constraint in the model design process used by any CAD software/application for 3D printing.

1.1 Motivation

The study of 3D printing and its applications have recently focused on emerging and disruptive technologies created by application of 3D printing. A closer observation of 3D printed objects shows few imperfections that are sometimes corrected with milling (machining). These imperfections are usually detected through physical measurements of dimensions of finished products. While this extra processing/correction step is required in most circumstances, it may be impractical in several applications. Every 3D printer has a quoted resolution in x, y and z dimensions. This work seeks to establish a framework for improving quality of 3D printed objects by ensuring that design features observe the quoted resolution of the applicable printing technology. As we experience application of 3D printing in professional areas including medical and electronics manufacturing, quality assurance gains significance. When 3D printing equipment manufacturers quote their printer resolutions, this information can serve as an input into a model to validate that the applicable printer will be able to produce critical features in the model of interest. The potential cost and time savings, gained by ensuring that CAD designers avoid features smaller than the printer resolutions, is quite significant.

Our work started with an investigation of the printability of muscle models using various 3D printing processes. We visited various 3D printing organizations and it was obvious there are still some challenges. The challenges were in the area of materials and printing modality and their constraints. It was during our investigation that we discovered that when models are printed with different printer settings, the resulting output varied greatly in terms of appearance and strength. Of most important was the fact that with very low resolutions, the joints that were supposed to allow movement fused. On interrogation, the technicians confirmed that this is due to the resolution. This can be proven by understanding that higher resolution mean much finer depositions and hence the ability to reproduce very small features. If a feature is an actual clearance between two objects, a finer resolution is more likely to achieve it than with a lower resolution.

We also looked at the iterative process of 3D printing which, to a great extent, relies on the expertise of the user. We considered this situation as drawback in the 3D printing workflow.

The above gave impetus to our thesis for suggesting that knowing the implication of any chosen resolution up front will save the iterative steps involved to achieve a particular desired result. If a technician can know by using our model that there will be defective regions, the technician will save production time and cost.

Our motivation also includes the packaging of our system as a plugin to existing 3D CAD design applications to assist in notifying/guiding the designer when they are about to perform shape operations that will results in features/regions in the 3D model that will be unprintable by the chosen printer. Our model can also serve as a plugin to the existing or new slicing applications to validate the slicing outputs to ensure that regions/features that are unprintable are flagged for correction. Our model provides information to assist in correcting imperfections in 3D model owing to features smaller than the printer resolutions.

1.2 Overview of Our System

The purpose of the research is to improve the quality of 3D printed objects in biomedical applications by detecting unprintable regions or preventing them before the slice file is sent to the 3D printer. We will apply a ray casting technique to boundaries of contours in slice sets of 3D data. With ray casting we are to compute and determine regions that are solids. These are regions that the applicable 3D (additive) printing method will have to fill with materials. We will then compare the sizes of the region with the resolution of the 3D printer. We will set regions with sizes lower than the resolution of the 3D printer as defective. The user can correct the regions by making them slightly larger than or equal to the resolution of the 3D printer, and then run the sample through our system again to confirm that those defects do not exist anymore. We will also compare the sizes of the spaces between regions and compare to printer resolution. A threshold for this gap, set by the user will be used to check regions. If the sizes of gaps between regions are lower than the set value, our model will flag the region as potentially defective.

Figure 1-1 shows the overview of our system. Upon reading 3D slice data generated by a slicing application, such as Slic3r[11], in simple Scalable Vector Graphics (SVG)[11] polygon sets, our system performs ray casting on each slice to determine solid regions. Then the length of the solid region is compared to the x resolution of the 3D printer. Regions/features on each contour in the slice having lengths less than the x resolution of the printer will be counted and presented to the user. The regions are also indicated with red markers on the slice sets. With varying printer resolutions, our system is able to indicate that when the resolution value is lower, lesser error regions are detected. This check is also performed in the y direction.

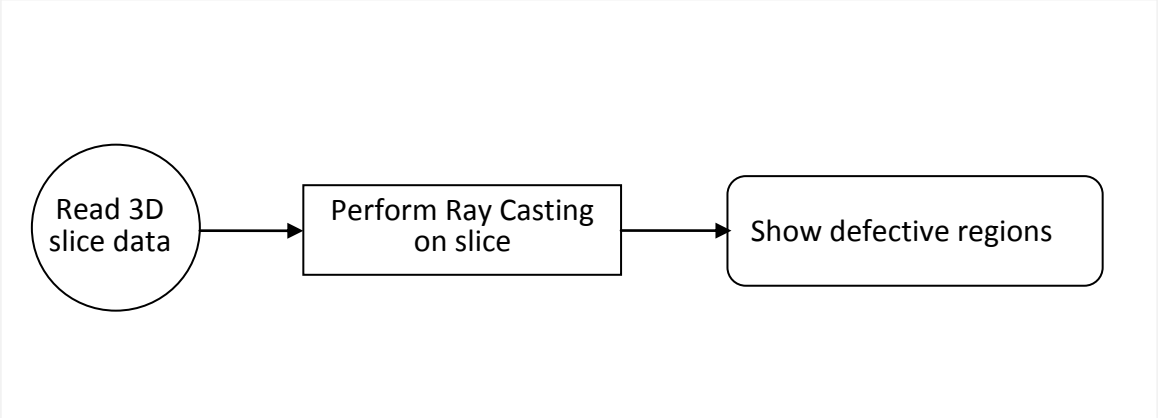


Figure 1-1: Overview of the system.

1.3 Our Contributions

1.3.1 Resolution Validation Method

Our investigation is in 3D space. The regions or features we are validating are occurring in 3D. However, we are presenting a simple method for performing this same validation using 2D CAD slicing dataset. The value of this method as alternative becomes evident when the performing the task in 3D is not feasible.

1.3.2 SlicingResValidator

Our web-based application, SlicingResValidator is hosted online and can be accessed at http://198.12.148.129:3002. It uses Thnigiview.js and d3.js for 3D rendering. It requires two set of files:

- An STL file for rendering of the 3D model
- An SVG slicing output (preferable from Slic3r) representing the layer-by-layer output of the 3D CAD model

1.3.3 Android STL 3D CAD Viewer and STL Parser

For visualization in mobile devices, we were also able to develop a C# component for parsing STL files (ASCII and binary). This was also ported into Android using Xamarin Mono (Java to .Net) framework.

1.4 Thesis Organization

This document consists of five chapters. The introduction in Chapter 1 provides a broad overview of the problem statement and the proposed solution. As we are dealing with 3D printing, in Chapter 2 we have provided a detailed review of the 3D printing technologies and some of its current applications. In Chapter 3, we linked the literature with our research by providing a methodology for our new system. In Chapter 4 we discussed the results we obtained and limitations when we ran a few samples through our system. In Chapter 5, we provided some conclusions and proposed applications of our new system. In appendix we provided a screenshot of our system as well as source codes implementing the core algorithms.

Chapter 2. Literature Review

2.1 Medical Applications of 3D printing

3D printing has been used in various medical applications including maxillofacial prosthetic rehabilitation [57][58], orthotic rehabilitation [59][60], dental technology[61], surgical applications[62] and in support of research projects [53].

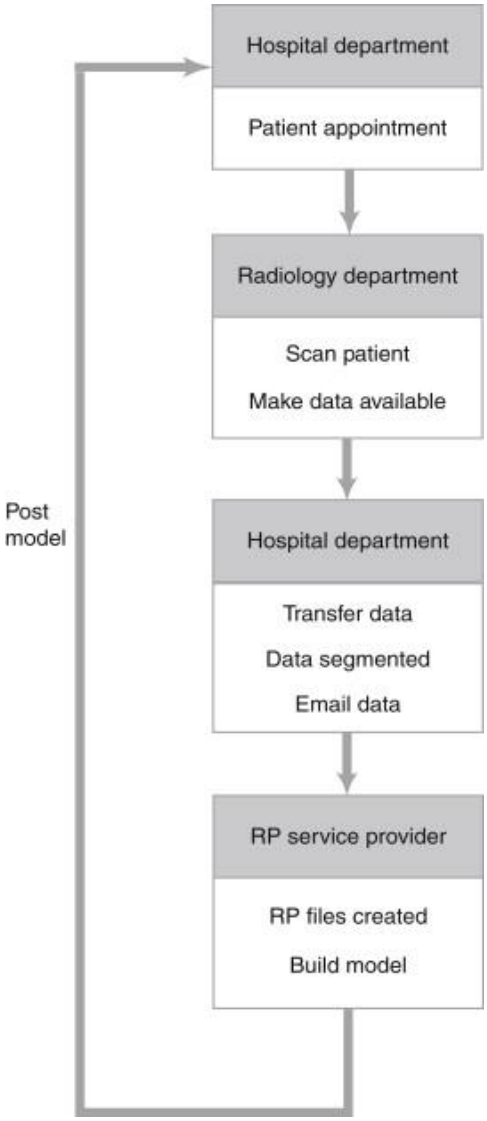
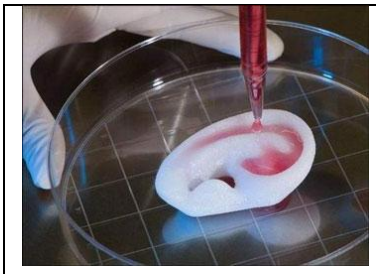
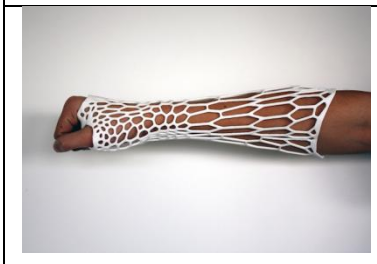


Figure 2-1: Rapid Prototyping model in a hospital (adapted from [53])

In Figure 2-1, a model for managing and maintaining integrated 3D printing and rapid prototyping capability is presented. The objective is to reduce cost, protect data, leverage clinical resources and reduce patient wait times. The model begins with a patient appointment where imaging data is acquired via magnetic resonance imaging, computed tomography, ultrasound or x-ray. This imaging data is presented in formats that allow for segmentation. Segmentation involves the partitioning of imaging data into contours with labels for further processing. The segmented data is converted into any of the 3D CAD formats and used to prepare the slice datasets used by the 3D printing software. This dataset is used to produce machine codes/instructions for making the physical model. Recent advancements in the area of clinical applications of 3D printing have been towards the ability to make parts quickly, validate accuracy of parts and receive feedback in little time.[36]

3D printing techniques has been applied to 3D tissue/organ development, bio-cell printing and creation of scaffolds for tissue engineering, and actual clinical applications for various medical parts [54][55][56] (see Figure 2-2).

	At Organovo, scientists were able to generate a few “human tissue samples of cardiac muscles, lungs and blood vessels using plastics, resin-based adhesives and other exotic materials” using 3D printing. (adapted from [63]).
	After scanning his arm for a rough outline with Xbox Kinect (c) Jake Evill created “Cortex Cast” – a honey comb cast for replacing traditional cast to provide support around broken bone [64]

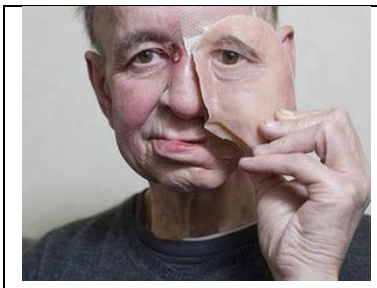
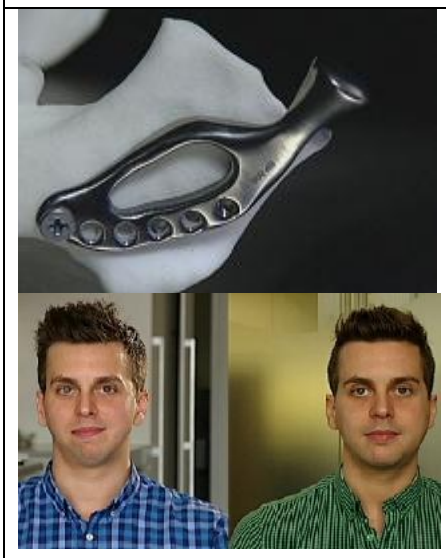
	A side of Eric Moger's face was removed after being diagnosed with an aggressive kind of cancer and a tumor the size of a tennis ball. A nylon facial shell was created using a mirrored image of the existing side of his face. An attached magnet keeps the plate in place to allow him to eat and drink properly again.[65]
	Titanium, 3D-printed prosthetic jaw implanted in Melbourne man in Australian first surgery[67]

Figure 2-2: Few examples of recent medical applications of 3D printing

2.2 Basic 3D Printing Process

3D printing (3DP) or rapid prototyping (RP) is an additive manufacturing process that involves the production of physical objects by adding thin successive layers of materials without using molds [13]. The models being printed can be obtained via image acquisition from mobile scanners [15], Magnetic Resonance Imaging (MRI), Computed Tomography (CT), positron emission tomography, direct 3D CAD models. This enhances the rapid prototyping process as the technology is capable of producing a near-net-shaped and multi-colored part. 3D printing is also referred to as Layered Manufacturing (LM) [13].

The 3D printing process, Figure 2-3, starts with a CAD data model generation. The data is "sliced" into successive layers. A slice is a collection of contours to be filled during printing.

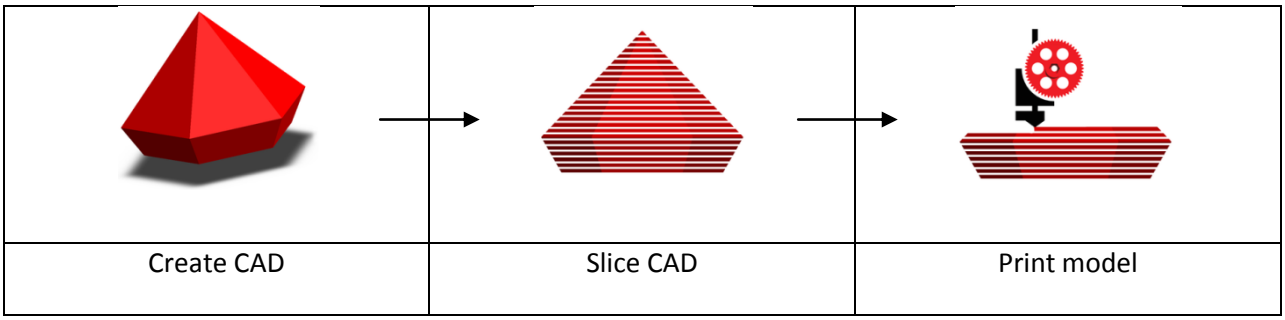


Figure 2-3: 3D Basic Printing Workflow [14]

2.3 3D Printing Methods

2.3.1 Fused Filament Fabrication (FFF)/Fused Deposition Model (FDM)

Developed by S. Scott Crump, Stratasys in Eden Prairie, in the late 1980s [7], FDM (Figure 2-4 and Figure 2-5) involves the laying down of materials, such as plastics or metals in layers. A thin wire of material is unwound from a coil and supplied to an extrusion nozzle which can turn the flow on and off. A worm-drive pushes the material through the nozzle at a controlled rate. The nozzle turns the flow on and off and is heated to melt the material. The nozzle is moved in the x-y-z direction. The build platform can also be moved in the z direction. The nozzle and/or the build platform movement is controlled by a computer-aided manufacturing (CAM) software package. The hardening of the material immediately after extrusion from the nozzle forms the part. In addition to thermoplastics, other materials used in FFF include ABS, polyamide, polycarbonate, polyethylene, polypropylene and casting wax.

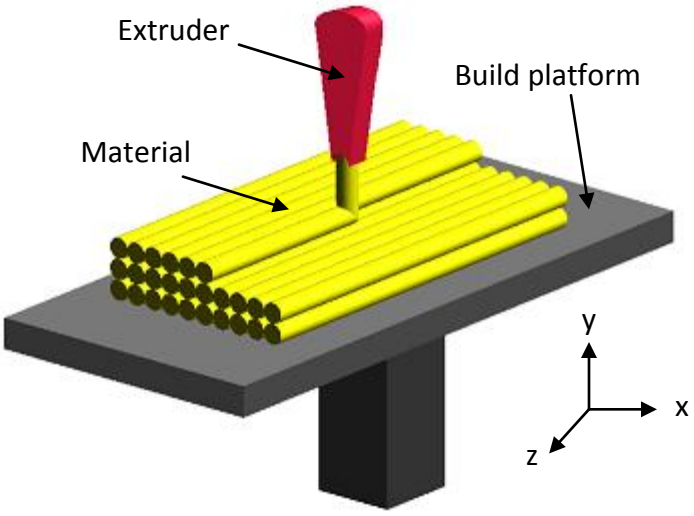


Figure 2-4: Fused Filament Fabrication (FFF)/Fused Deposition Model (FDM) Process [16]

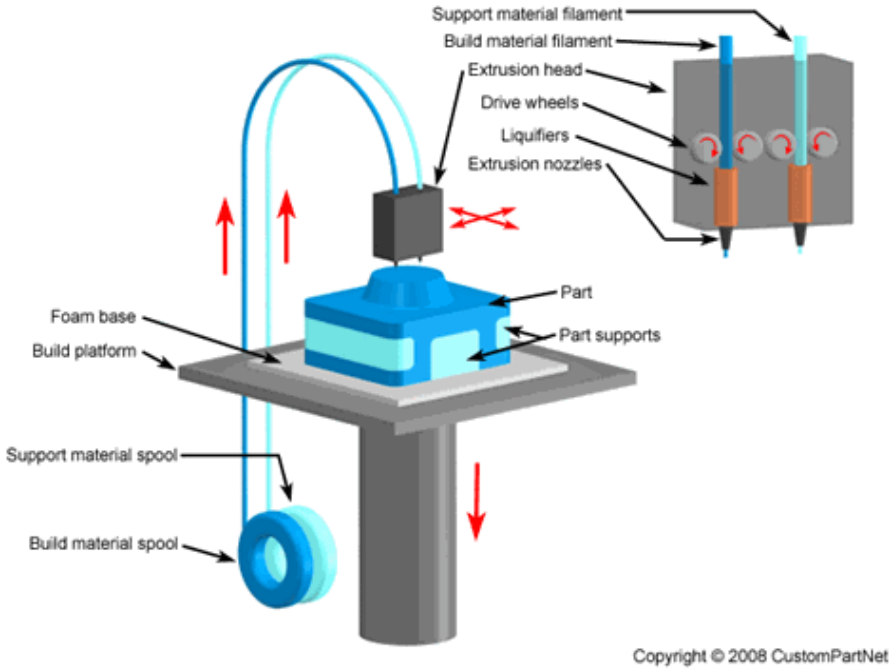


Figure 2-5: Fused Filament Fabrication (FFF)/Fused Deposition Model (FDM) Process 2 [17][19]

2.3.2 Electron Beam Freeform Fabrication (EBFF)/Electron Beam Melting

Developed by Arcam AB in Sweden [19], EBFF involves the use of a focused electron beam in a vacuum to create a molten pool from metal powder. In Figure 2-6, the beam is moved with respect to the surface of the substrate. Metal wire is fed into the molten pool and the deposit solidifies immediately after the electron beam has passed. The parts produced from EBFF are usually fully dense, void-free, and extremely strong.

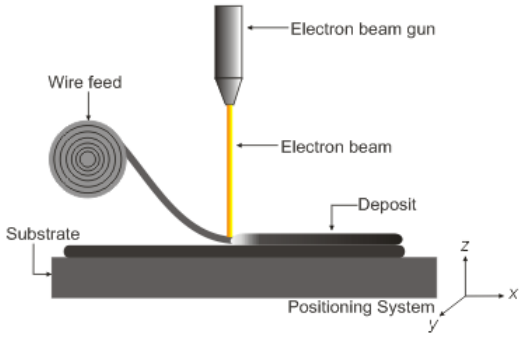


Figure 2-6: Schematic of Electron Beam Freeform Fabrication/Electron Beam melting (adapted from [19])

2.3.3 Stereolithography (SLA)

SLA was developed and patented in 1986 by Charles W. Hull [10]. It involves the making of solid objects by successively applying thin layers of an ultraviolet curable material on top of each other. A liquid photosensitive polymer (resin) is solidified in ultraviolet light following a pattern defined in the CAD model (see **Figure 2-7**).

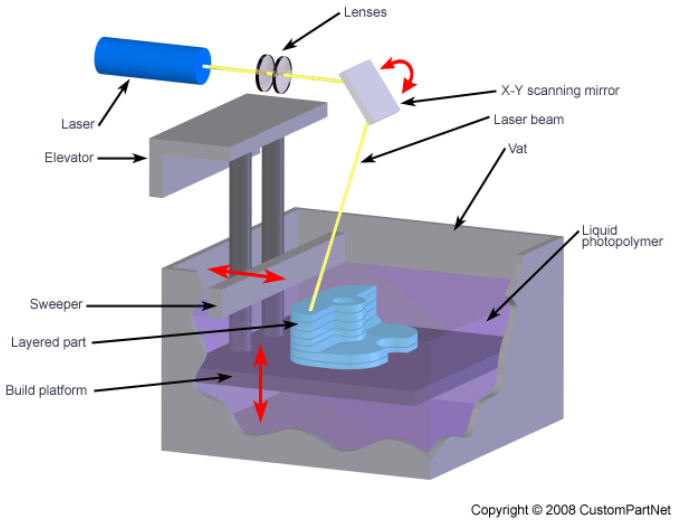


Figure 2-7: Stereolithography (SLA) (adapted from [20])

2.3.4 Selective Laser Sintering (SLS)

Developed and patented by Dr. Carl Deckard and Dr. Joe Beaman at the University of Texas, Austin [8], SLS involves the melting of materials below liquefaction by a laser beam, following a path defined in the CAD model to form the desired object. No wire or filament is involved. Rather a powder bed containing the sintered material is used (see Figure 2-8).

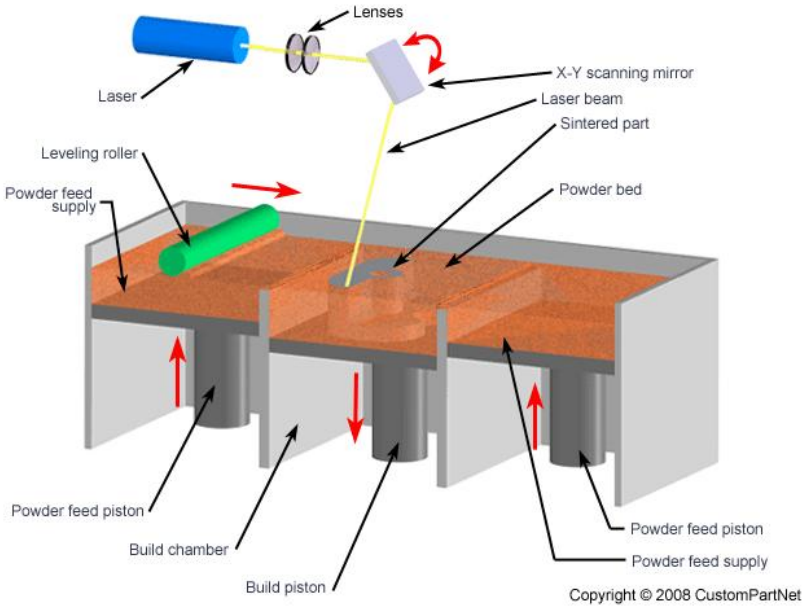


Figure 2-8: Selective Laser Sintering (SLS) [23]

2.3.5 Direct Metal Laser Sintering (DMLS)

Rapid Product Innovations (RPI) and EOS GmbH jointly developed DMLS in 1994. DMLS is similar to SLS. DMLS is the first commercial rapid prototyping method to produce metal parts in a single process [9]. In DMLS, complete melting of metal powder (20 micron diameter), free of binder or flux agent, is achieved by scanning of a high power laser beam to build the desired part with properties of the original material. "DMLS has a higher resolution and is therefore capable of more intricate shapes than SLS due to the use of thinner layers enabled by a smaller powder diameter" [9] . Medical implant and aerospace parts for high heat applications are usually fabricated with DMLS. A powder bed or powder deposition can be used in DMLS. Powder methods use only one material while powder deposition can use more than one material (Figure 2-9).

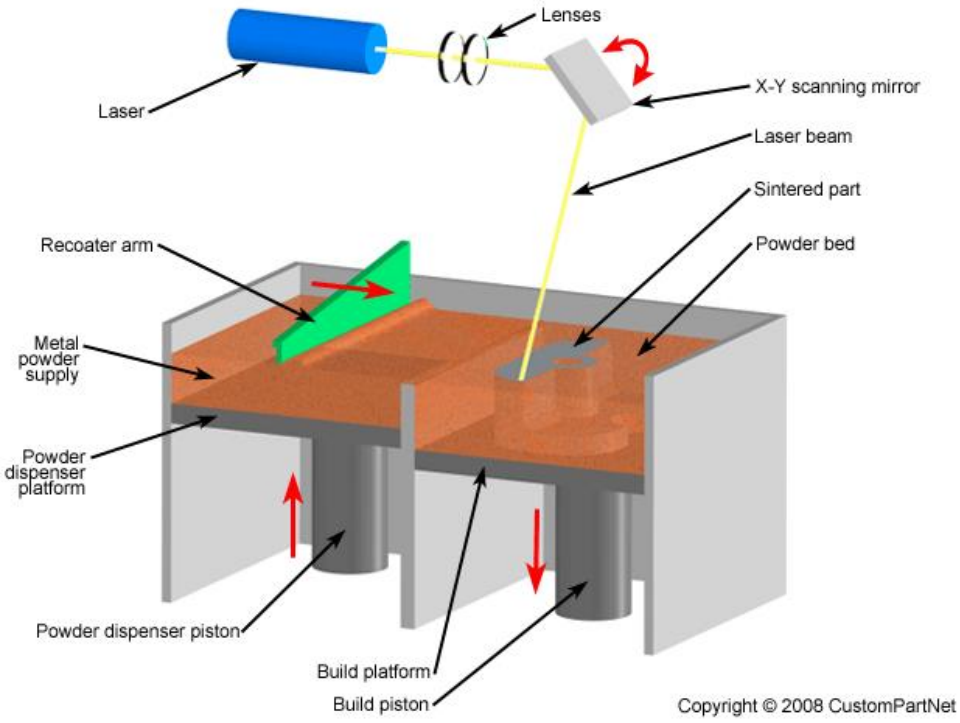


Figure 2-9: Direct Metal Laser Sintering (DMLS) [9][26]

Table 2-1: Comparison of Capabilities of 3D Printing [22]

Method	**Material type**	**Materials**	**Surface finish**	**Build Speed**	**Applications**
Direct Metal laser Sintering (DMLS)	Powder (Metal)	Ferrous metals such as Steel alloys, Stainless steel, Tool steel; Non-ferrous metals such as Aluminum, Bronze, Cobalt-chrome, Titanium; Ceramics	Average	Fast	Form/fit testing, Functional testing, Rapid tooling, High heat applications, Medical implants, Aerospace parts
Electron Beam Melting (EBM)	Solid (Freeform fabrication)	Titanium alloys	Strength, elasticity, fatigue, microstructure	Fast	Stainless steel, Titanium, Aluminum, Hard metals, Copper, Niobium
Selective Laser Sintering (SLS)	Powder (Polymer)	Thermoplastics such as Nylon, Polyamide, and Polystyrene; Elastomers; Composites	Average	Fast	Form/fit testing, Functional testing, Rapid tooling patterns, Less detailed parts, Parts with snap-fits & living hinges, High heat applications
Fused Deposition Model (FDM)	Solid (Filaments)	Thermoplastics such as ABS, Polycarbonate, and Polyphenylsulfone; Elastomers	Rough	Slow	Form/fit testing, Functional testing, Rapid tooling patterns, Small detailed parts, Presentation models, Patient and food applications, High heat applications

Stereolithography (SLA)	Liquid (photopolymer)	Thermoplastics (Elastomers)	Smooth	Average	Form/fit testing, Functional testing, Rapid tooling patterns, Snap fits, Very detailed parts, Presentation models, High heat applications

2.4 3D Printing CAD Data

The process planning of additive manufacturing, as shown in Figure 2-10, begins with the creation of the CAD model. This can be done with any of the popular 3D applications. It can also be obtained from 3D medical imaging data. 3D files can be exported into any of the following formats: STL (Standard Tessellation Language) [24][25][28][29], IGES (Initial Graphics Exchange Specification), STEP (Standard Graphics Exchange Format) [30], PLY (Polygon File Format)], OBJ (Wavefront's Object File), etc. Nearly all CAD applications are able to export and use STL formats. For the purpose of our research, we have chosen STL file format. Since STL has robust documentation and literature available, it is non-proprietary, and covers most of the key 3D modeling data elements needed to support 3D printing, we consider it useful to work with it in our research. However, there are also tools and applications for converting CAD models from one format to the other. The choice of STL as a file format for discussion is not an endorsement of the standard.

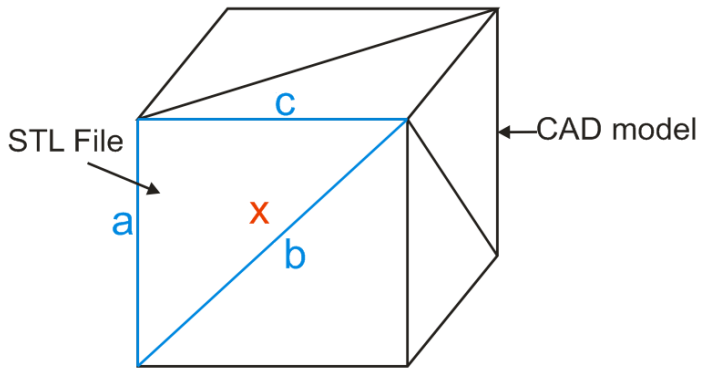


Figure 2-10: Standard Tessellation language (STL) File and CAD model [24]

STL is developed by 3D systems [24] and is supported by many CAD applications. It describes the surface geometry of three-dimensional objects without any representation of color, texture or other common CAD model attributes. There are two formats for STL file: ASCI and Binary. Binary files are smaller and more compact. Hence they are more common.

STL describes solids with triangles and their unit normal. Each triangle is described as a set of three vertices (ordered by the right-hand rule) of the triangle using a three-dimensional Cartesian coordinate system. STL does not contain negative numbers and scale information. No unit is adopted.

```
solid name
```

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

```
endsolid name
```

Figure 2-11: ASCII Standard Tessellation language (STL) File format

```
UINT8[80] – Header
UINT32 – Number of triangles

foreach triangle
REAL32[3] – Normal vector
REAL32[3] – Vertex 1
REAL32[3] – Vertex 2
REAL32[3] – Vertex 3
UINT16 – Attribute byte count
end
```

Figure 2-12: Binary Standard Tessellation language (STL) File format

The ASCII STL format begins with the line **solid** and an optional **name** attribute (Figure 2-11). The binary format (**Figure 2-12**) does not contain the **solid** line but begins with an 80 character length unsigned integer bytes. Notable information in the STL file is the triangle definition. It contains all the triangles each described with a normal and three vertices.

The facets (**Figure 2-13**) define the surface of a three-dimensional object. The orientation of the facet is always out (i.e. facing the direction of the unit normal). The direction of the facet is outward and the vertices are listed in counterclockwise order when looking at the object from outside (right-hand rule) (Fig. 2-9)

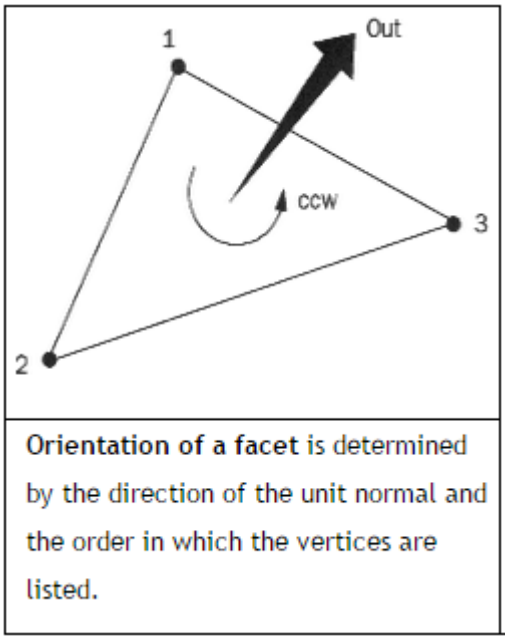


Figure 2-13: STL Facet Normal [31]

Recently, a variation of STL file is being created to include color information **[27]**. This is to support the evolving applications of 3D printing that is including color components. Color helps to make the printed object more realistic. Some software vendors use the two bytes for "attribute byte count" at the end of every triangle to store a 15-bit RGB color:

- bits 0 to 4 are the intensity levels for blue (0 to 31),
- bits 5 to 9 are the intensity level for green (0 to 31),

- bits 10 to 14 are the intensity level for red (0 to 31),
- bit 15 is 1 if color is valid, or 0 if the color is not valid

Other software applications use the 80-byte header at the top of the file to represent the overall color of the entire part.

When defining triangle in STL files, it is recommended to obey the **vertex-to-vertex rule [31].** This rule states that each triangle must share two vertices with each of its adjacent triangles. Hence, no triangle should have a vertex lying on the side of another (adjacent) triangle.

The next step in the 3D printing workflow is the creation of the slicing output. CAD applications used in 3D printing translate the STL files into machine instructions (G-Codes) for use by the applicable printers for making the physical part.

2.5 3D Printing Slicing and G-Code

2.5.1 Slicing

O. Topcu, Y. Tascioglu and H.O Unver[32] presented a method for slicing CAD Models for the purpose of developing G-Codes. This involves the cutting of the triangulation surfaces (facets) into shapes with heights equal to that of the slice thickness. These lines are then joined to form contours which are used for tool path data generation (G-Codes). There is presently no industrial or formal specification for slice data.

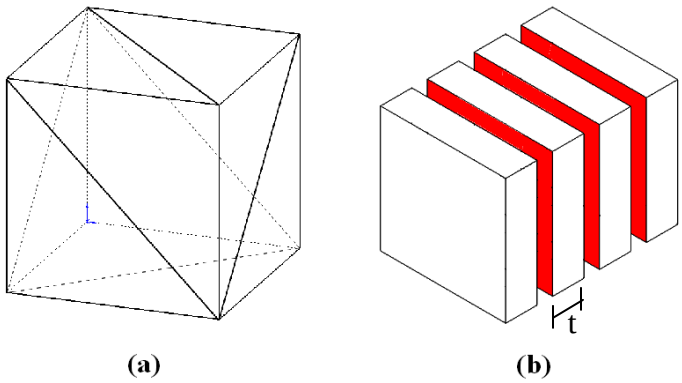


Figure 2-14: (a) Tessellated cube with 12 facets (b) a sliced cube [32]

An algorithm for slicing 3D models developed by Topcu et al [32] uses STL file as input. In **Figure 2-14**(a), the cube is tessellated into 12 triangles in the STL. The cube is cut into slices in **Figure 2-14**(b). The goal of the slicing algorithm is to produce contours for generation of G-Code. If the slicing thickness (t) is larger, the likelihood of facets falling in between the slices increases. This will make those facets not to be sliced. To avoid this, the thickness of the slices is reduced. This gap between adjacent slices defines the layer thickness. The layer thickness is a variable in the slicing algorithm.

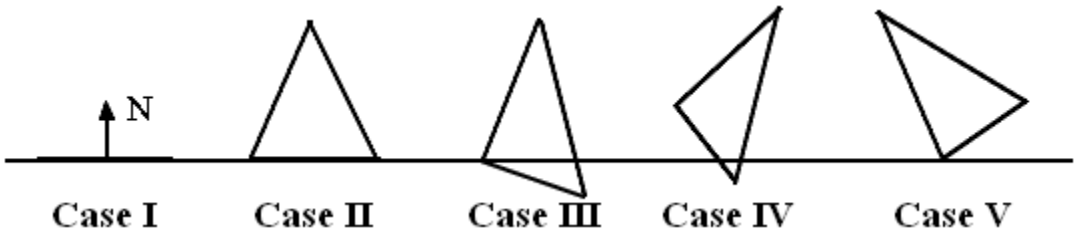


Figure 2-15: Possible intersection cases [32]

The algorithm considers five possible cases for intersection of the facets with the slicing line. The part slicing is purely a geometrical process. In **Figure 2-15**, the possible cases of intersection are shown. Representing a facet with its vertices as $V_{1,2,3}(x_{1,2,3}, y_{1,2,3}, z_{1,2,3})$ and its normal vector as $N(x_w, y_w, z_n)$, the algorithm presents the mathematical relationship for the possible intersections as:

I. $z_1 = z \,\&\, z_2 = z \,\&\, z_3 = z$

II. $z_1 = z \,\&\, z_2 = z \,\&\, (z_3 < z \,||\, z_3 > z)$

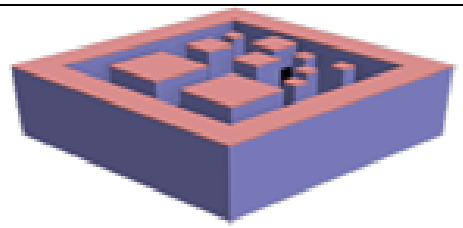
III. $z_1 = z \,\&\, ((z_2 < z \,\&\, z_3 > z) \,||\, (z_2 > z \,\&\, z_3 < z))$

IV. $((z_1 < z) \,\&\, (z_2 > z \,\&\, z_3 > z) \,||\, (z_1 > z) \,\&\, ((z_2 < z \,\&\, z_3 < z))$

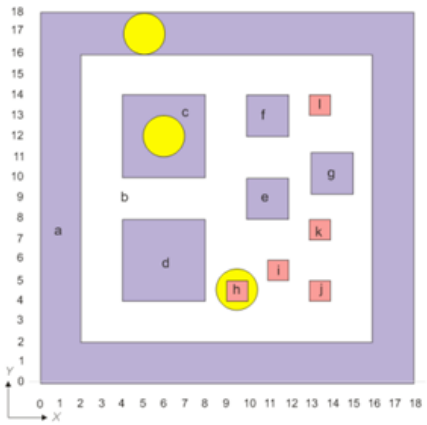
V. $z_1 = z \,\&\, ((z_2 > z \,\&\, z_3 > z) \,||\, (z_2 < z \,\&\, z_3 < z))$

A slice is a collection of contours. Each contour is a cross-section of the part at the slice index. This model, while aiming to present an open source framework for slicing 3D data and consequently generating G-Code, as part of the process planning steps in the 3D printing

workflow, did not focus on the implications of resolution of the 3D printers on the quality of the output.



(a)



(a)

Figure 2-16: (a): A model (blue) with one slice (red) (b): Slice and its contours

The slice in the model in **Figure 2-16**(a) is presented as a collection of contours in (b). This data can be modeled in hierarchical structure.

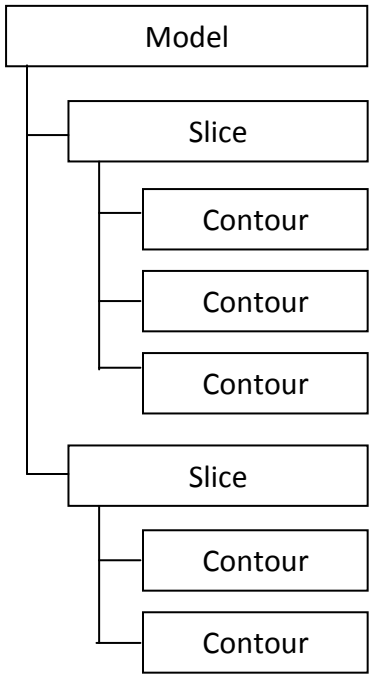


Figure 2-17: Slicing data structure

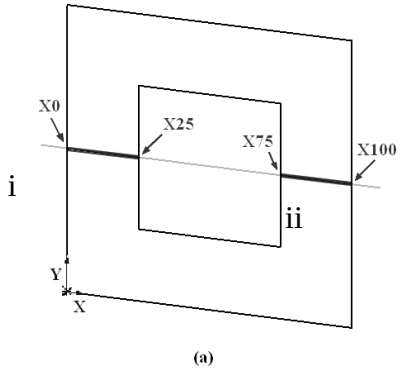
In **Figure 2-17** a model is presented as a collection of slices. A slice is a collection of contours. A contour is a collection of shapes or lines. Contours are key to generation of tool paths or G-Code.

Various applications exist for slicing of 3D data and their outputs can be used by 3D printers to build objects of interest. A search for slicing applications from popular search engines returns few popular slicing applications that are both free and commercial. They include:

- Cura (open source project by Ultimaker)[47]
- KISSlicer (commercial project) [48]
- Skeinforge (open source project) [48]
- Slic3r (open source project) [48]
- RepSnapper (open source project) [49]
- Miracle-Grue/Makerware (freeware project by Makerbot) [50]

2.5.2 G-Code Generation

G-Code (G programming language) [34] is used to tell the 3D printer how to make the part. It is also a common language for numerically controlled machines (of which 3D printer is one). In relation to the slicing sets, the contours provide boundaries for the deposition of materials layer by layer until the entire part is built.




```
G00 X0.0 Y50 Z1.0 S2000.0 T10
G01 Z-1.0 F200.0
G01 X0 Y50 F300
G01 Z1.0 F200
G01 X25 Y50 F300
G01 Z-1.0 F200.0
G01 X75 Y50 F300
G01 Z1.0 F200
G01 X100 Y50 F300
G01 Z-1.0 F200
G01 X100.0 Y50 F300
```

(b)

Figure 2-18: (a) Schematic illustration of G-Code generation and (b) resultant code of the algorithm [32].

In **Figure 2-18**(a), the dark lines (i) and (ii) represent a tool path for material deposition. The G-Code in **Figure 2-18**(b) is the instructions for the machine to do the deposition.

The code snippet in **Figure 2-19(b)** includes some popular G-Code commands. Below is a description of few G-Code commands:

G00 X0.0 Y50 Z1.0 S2000.0 T19

G00 is the first statement in this command line. It states that this is a command for movement.

X0.0 Y50 Z1.0 is the second information in this command line and serves as the arguments for the move command above. It is instructing the machine to "move the X in 0 millimeters to the left, the Y 50 millimeters forward and the Z 1 millimeters downward."

S2000.0 is a command parameter for instruction sent to the motor. It can be time in seconds; temperatures; or voltage

T10 is a tool selection instruction. Tools can be extruders for a 3D printer.[68]

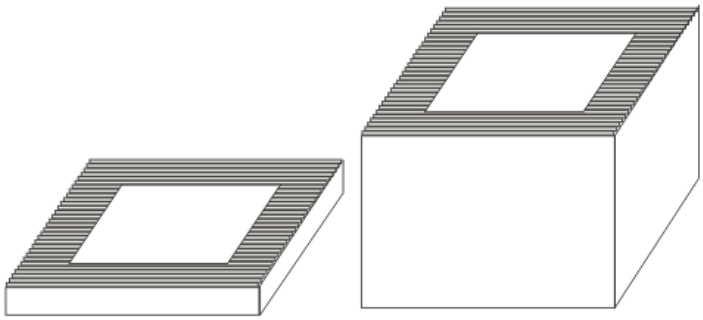


Figure 2-20: Build stages on executing G-Code

In **Figure 2-20**, as the G-Code is being executed, the part is built in successive layers. The time taken to build the entire part depends on the following factors:

- The speed of the moving nozzle
- The deposition volume
- The thickness of the slice

The above variables are also influenced by the material type and 3D printing process used. The G-Code provides these values to the 3D printer.

2.6 Slicing and 3D Printing Quality Assurance

Baumann F. et al [36], proposed a framework for achieving a comparable quality assessment of both slicing tools for FDM printers and FDM printers themselves. The framework chose a few popular slicing tools based on their reliability (ability to handle all test models), G-Code compatibility and application configurability. The properties being configured included: print temperature; print bed temperature; layer thickness; fill density; print speed and minimum layer print time.

Baumann's work performed a few tests on 3D printed models with the intention of evaluating the influence of the slicing process used in the G-Code generation. The work also provided an overview of the model and why they are chosen. The work also attempted to define root causes of detected defects in 3D printing, the main goal is to produce models that are close representations of the actual articles being reproduced. A few factors/constraints make it difficult for the 3D printing method to achieve this objective. Overhangs and bridges, plastic remains, strength etc, are a few limitations that are imposed by the printing method as well as material [36]. Baumann's work focused on evaluation of 3D slicing applications and this evaluation is based on the test of these attributes. Overhang tests were performed with and without fan cooling and the print output of the GCodes generated by each slicing application is evaluated.

Text features in 3D CAD models were also tested in Baumann's work. Text tests exposed difficulties with round shapes of few letters especially as they get smaller. Intersections were found to also present difficulties.

A precision test is also proposed by Baumann's work deriving from the knowledge gathered from the various tests. In the test model, intricate shapes and fragile geometric models are being evaluated for the G-Codes generated by different slicing applications.

Baumann's work was able to establish that KISSLicer and Slic3r handle fine structures better than other slicing applications. The work established that the choice of slicing tool has a huge influence on the quality of overhangs and bridges. Slic3r is shown to have a good support for bridging but Cura had the best support for overhangs. The work also established that using cooling fans to cool the printed model can improve the quality of overhangs and bridges significantly.

Baumann's work also proposed, as a future work, the automated testing for quality assessment of the identified features in order to provide faster and more accurate results. We are looking in that direction but consider the influence of printer resolution. Practically all 3D printers, just like cameras, quote resolutions but in 3D space (x-y-z). A consideration of this resolution in determining the ability of a printer to accurately print a model should be an integral component of a slicing application. When feasible, an add-on, plugin or external application should be used to evaluate a slicing output against the printer resolution. Given that the slicing is performed after modeling, and any defects will be corrected by going back to the model, an inclusion of a resolution checking process, during design, will save time and cost and further improve the quality of printed outputs. When vendors quote printer resolutions, evaluation of printers for their suitability for a given model can make use of a resolution-aware slicing process to select the best printer. The application of resolution based constraints to shape operations is implemented in Seagull Scientific's Bartendar software [69]. Bartender is a 2D application for designing and printing labels for personal and industrial/commercial use. When creating a new label, the application prompts user to select printer. The resolution and other machine configurations of the printer are used by the application to visually present the user with simulated outputs of their work. The benefit of this is seen where a user attempts to shrink a feature below the threshold allowed by the printer, this operation is immediately restricted. On investigation of 3D CAD applications such as 3D Studio Max, AutoDesk etc., we observed that there is no such enforcements or intelligence. The reason is quite obvious as the applications are not usually for G-Code generation. Slicer applications, such as Slic3r and Cura handle G-Code generation. If our method is to be applied in the 3D CAD design tools, the following flow will be applicable for mimicking the enhancements found in Seagull's Bartender:

Step 1: Launch application

Step 2: Select Printer or define core printer parameters such as X,Y and Z resolutions

Step 3: Load or design 3D model

Load: Check 3D CAD model for regions with potential defects (unprintable) due to printer resolution

Design: Perform test for each feature (in memory) as user is performing shape operations

2.7 Resolution-Aware Slicing

2.7.1 Printer Resolution

In 3D printing terms, resolution is the number of individual voxels that can be deposited in a given unit volume. It is usually express as layer thickness (z) and x-y resolution in dots per inch (dpi) or micrometers (µm). The layer thickness varies by printer and/or printing technology. For instance, printers involving lasers provide very high resolution. Typical layer thickness is around 100 µm or 250dpi. Some machines are capable of printing layers as thin as 16 µm (1,600 dpi) [22]

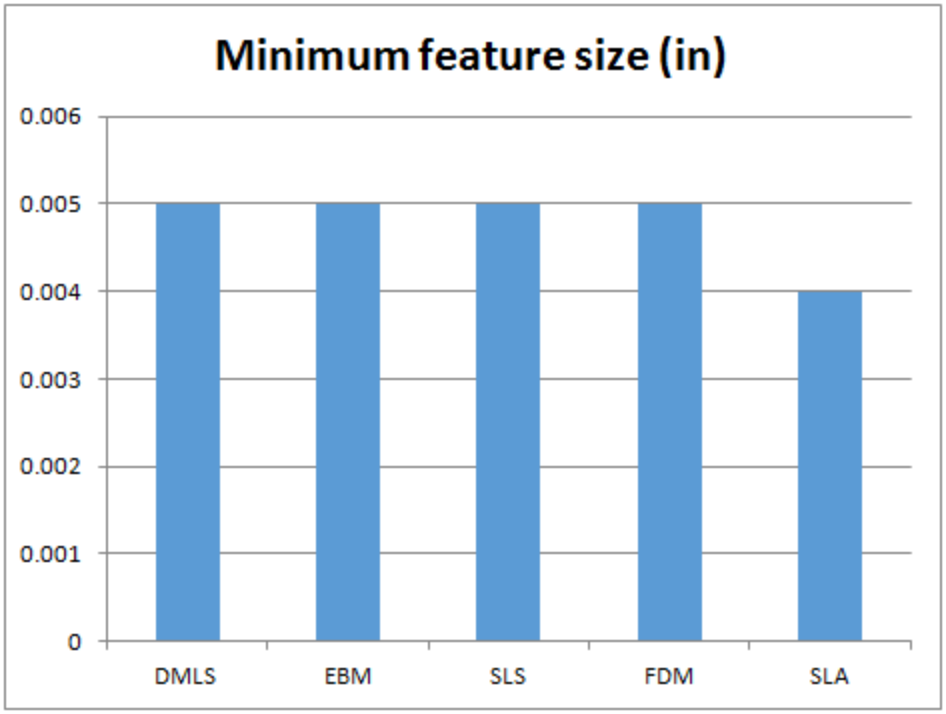


Figure 2-21: Comparison of Minimum Feature Size (adapted from [22])

We compared the data for five 3D printing technologies based on their capabilities as it relates to factors that affect print resolution. The factors include minimum feature size) and minimum layer thickness.

In **Figure 2-21**, direct metal laser sintering (DMLS), electron beam melting (EBM), selective laser sintering (SLS) and Fused deposition model all averaged a minimum feature size of approximately 0.005in. If the minimum feature size is less than the print resolution, it will not be possible to print the part. While this is not too common with the current application of 3D printing, it is imperative to take into consideration the fact that 3D printing is growing in application as newer uses of 3D technology is being investigated. Stereolithography (SLA) offered the best opportunity in terms of minimum feature size as it is capable of 0.004in feature size.

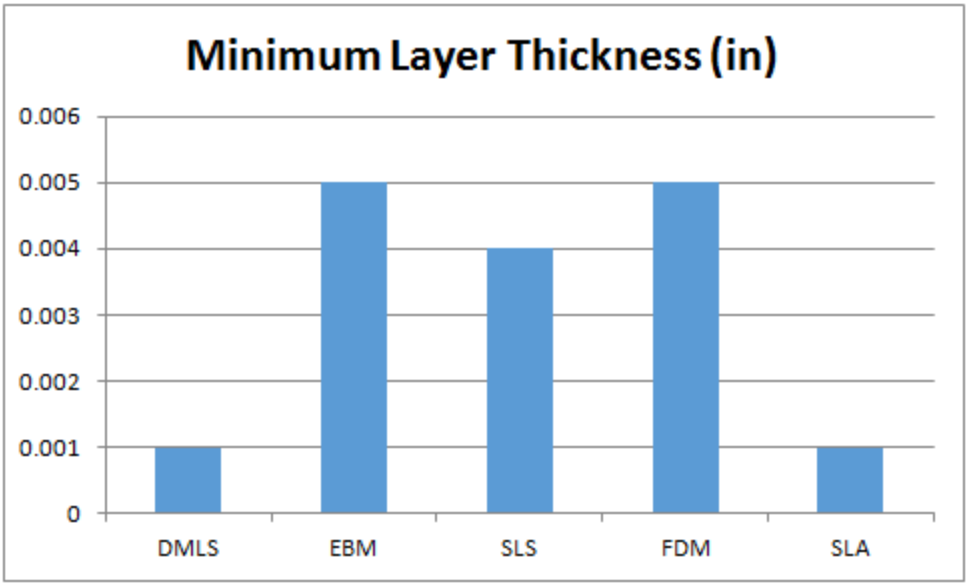


Figure 2-22: Comparison of Minimum Layer Thickness (adapted from [22])

The layer thickness has negative effect on the quality of the print work. As the layer gets thicker it becomes increasingly difficult for the printer to accurately target and print intricate shapes. In **Figure 2-22**, DMLS and SLA offered the best opportunities as they are capable of layer thickness of 0.001in. SLS is next with a minimum layer thickness of 0.004in while EBM and FDM offered the poorest performance with both capable of minimum layer thickness of 0.005in.

The factors that influence printing resolution include:

- Print accuracy along the x, y and z axes. The precision of the print head is determined by the x and y axis resolution. The z axis, where the layers are applied, determines how fine they will be.
- The viscosity of the binding agent
- The accuracy of color application
- Treatment of the 3D printed object after it comes out of the machine.

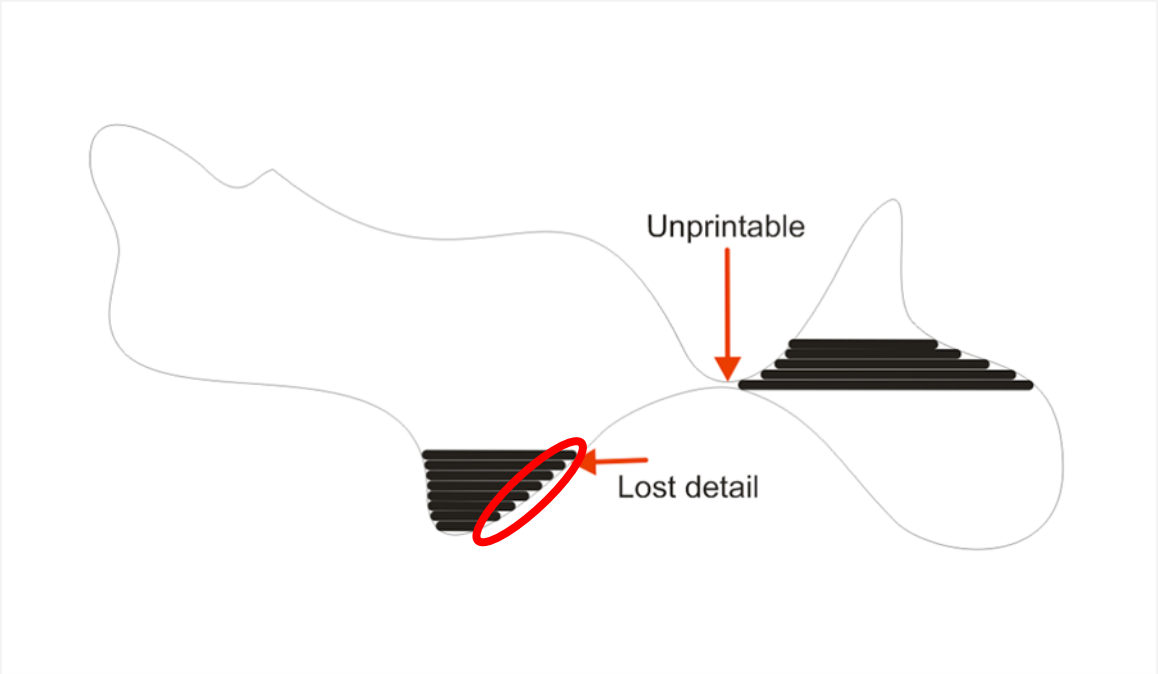


Figure 2-23: Effect of resolution on printability of features

In **Figure 2-23**, it can be shown that as feature/region sizes get smaller than the resolution of the printer the feature/region becomes unprintable by that printer. Also, the staircase effect common with 3D printing is also shown. The decreasing of layer thickness reduces the staircase effect. This can lead to long manufacturing time and high cost. A review of methods for slicing 3D data to handle staircase effect [44] identified a few methods including:

- Cusp height concept [37]
- Stepwise uniform refinement [38]
- Local adaptive slicing [39]
- Accurate exterior and fast interior [40]
- Efficient slicing method [41]
- Non Uniform cusp heights [42]
- Consideration of parabolic build [43]

All the methods above are used in adaptive slicing. The ultimate goal is to achieve optimal fineness (finishing) for curved or diagonal surfaces by increasing the number of layers and reducing layer thickness.

Feature detection algorithms utilize the concepts of line intersections. Rays are represented as straight lines shooting outward from source. The boundaries of a feature are the points where the lines intersect with the features. These boundaries when "stitched up" constitute a contour. A slice (layer) is made up of contours. We took advantage of the existence of these contours to propose that in a layer, we can determine the length and width of features, in 2D space, and compare them to the resolution of the printer. This comparison is what we present as the verification of the printer's ability to accurately produce the part. The same method can be applied in any direction by mere translation. So, for the purpose of this research, we are taking insights from the concepts of lines intersection.

2.7.2 Intersection of two lines and calculation of feature sizes

Two lines L_1 and L_2 on the same 2-dimensional plane can intercept. With L_1 being defined as (x_1, y_1) and (x_2, y_2) and L_2 being defined as (x_3, y_3) and (x_4, y_4) [[71]]. The intersection point (P) of line L_1 and L_2 can be defined using determinants:

$$P_x = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}} \qquad P_y = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}$$

The determinants can be written out as:

$$(P_x, P_y) = \left(\frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}, \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \right)$$

Ray casting (as demonstrated originally by Arthur Appel in 1968) [70] is a method employed to solve several computer graphics and computational geometry problems. It relies on ray-surface intersection tests. Techniques and problems solved using ray casting include:

- Problems involving determination of first object intersected by a ray (line)[45]
- Hidden surface removal based on finding the first intersection of a ray cast from the eye through each pixel of an image
- Ray tracing rendering algorithm that only casts primary rays in non recursive manner, or
- Volume ray casting which is a direct volume rendering method (volume ray casting), by "pushing a ray through" an object and sampling the 3D scalar field of interest along the ray inside the object. [47]

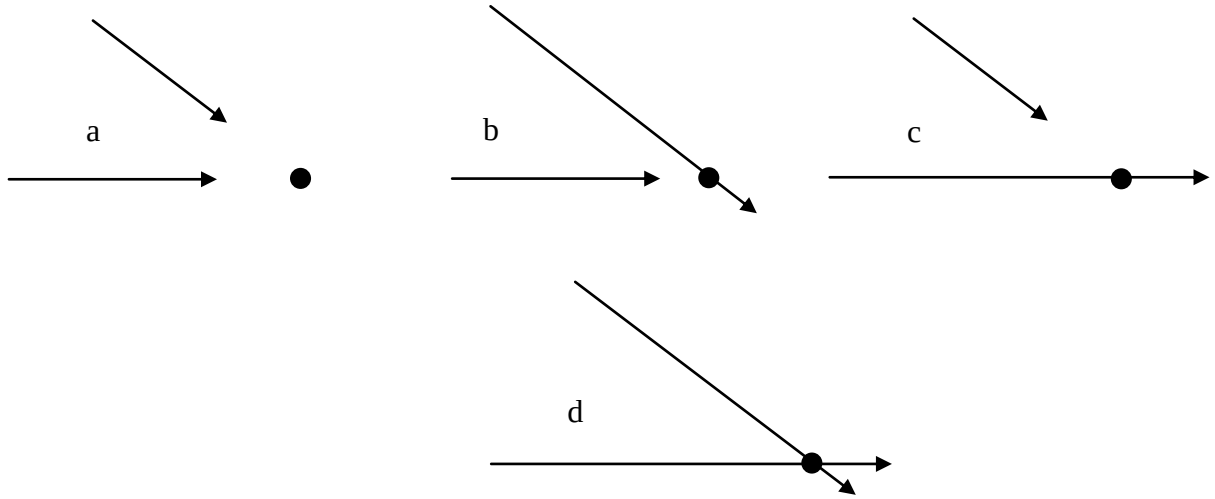


Figure 2-24: Possible intersection of 2 lines.

In the context of feature detection, we will be interested in a line (ray) striking the edge of a contour in slicing data. In **Figure 2-24**, we present all scenarios where two lines intersect. Only the points with the kind of intersections in **d** are used in our ray casting function. Furthermore, we are not interested in the scenario where two lines that are not parallel can intercept. We are interested in when they actually intersect and where the intersection occurred.

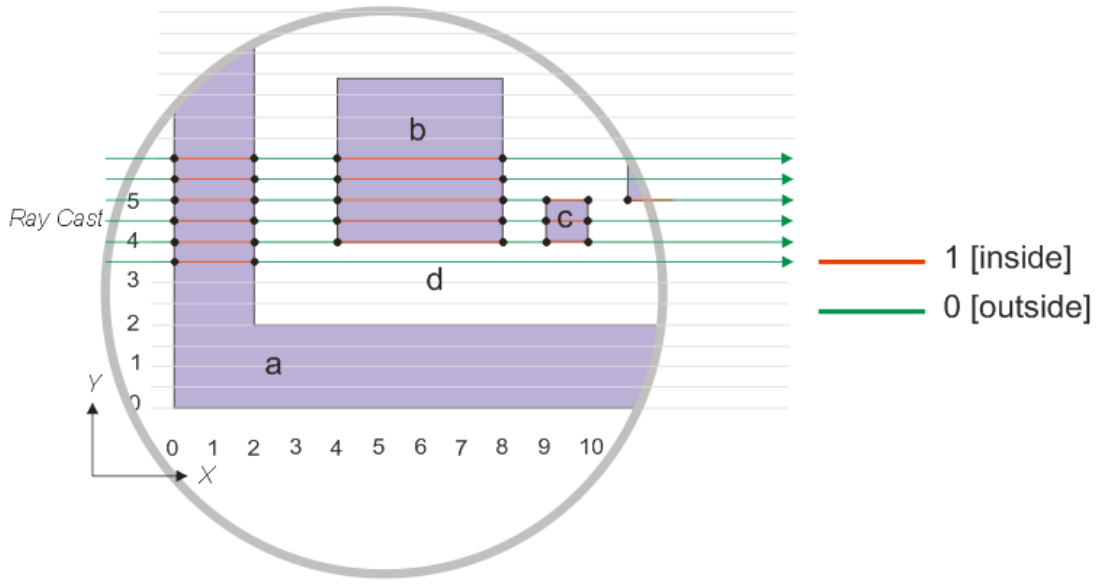


Figure 2-25: Schematic of Ray casting to detect solid features for size comparison.

In **Figure 2-25**, rays/lines were shot from outside a slice having a set of contours including a, b and c. The areas in gray a, b and c are solids. These rays intersect the edges of the contours at the points indicated with small black dots. For a line, the sections between points indicating solid regions of the contour are marked in red while the sections between points outside the contours are marked in green. Another annotation is in the form of even-odd designations or 0,1,0,1,0,... Where 0 indicates non-solid sections of the line and 1 indicates solid sections of the line. With the above, it is possible to determine an approximate size in terms of length of the feature/region along the ray (line). A comparison of this line to the printer resolution in the applicable direction (x or y) can be used to determine if the feature is printable or not using the particular printer. This is the focus of our work.

There is currently no application or solution addressing the problem. The reason is that the information needed to perform the computation in 3D space will be in memory during shape operations in any 3D CAD application. We are utilizing this simple ray casting method to solve this problem to prove that the solution is feasible in 2D space.

Other assumptions for this process include the following:

- Z resolution is equal to the printer layer height or frequency

- Solving the solution in one direction (say x) is sufficient since the solution can be applied via translation to serve for the other direction (y is x translated 90 degrees).
- The gaps between features or regions can be detected by applying same logic to lines falling on even nodes. If solid is an odd node in the ray casting designation, gaps will be even nodes and vice-versa.
- The model should be able to check with rays (lines shot diagonally. This will also be handled by translation. So, our model can be extended to include an extra parameter for angle of ray. An implementation of our model can use randomly chosen angles or request for this input from the user.

Chapter 3. Methodology and system

3.1 Objectives

The objective of this research is to improve the quality of 3D printed objects, by detecting upfront, potential defects as regions or features that will be unprintable due to printer resolution. Ray casting algorithm mentioned in section 2.5.2 will be applied to slicing output for the applicable printer. Each slice will contain a set of contours. The ray casting method will be used to determine the regions of the contours that are actual solids and the ones that are gaps. The length of the line between one edge of the solid portion of the contour and the other will be compared with the printer resolution in that direction. Regions having lines lower than the printer resolution in a particular direction will be flagged as having potential defects on printing. Same logic will be applied to gaps.

3.2 Data structure for Managing Slicing Data

A data structure is created to hold slicing data in memory for the subsequent computations. As shown in Figure 3-1, each slice is identified by **id.** A slice holds a collection of **contours**. Each **contour** is a collection of **lines**. A **line** will have an **id, x1, y1** (start point) and **x2, y2** (end points) (see **Figure 3-1**).

Ray casting will be performed for each slice. The ray casting intersection points will not be kept in memory to reduce computational load. Meanwhile, all the lines in a slice, having lengths lower than the defined resolution in the applicable direction will be kept in a list for results presentation.

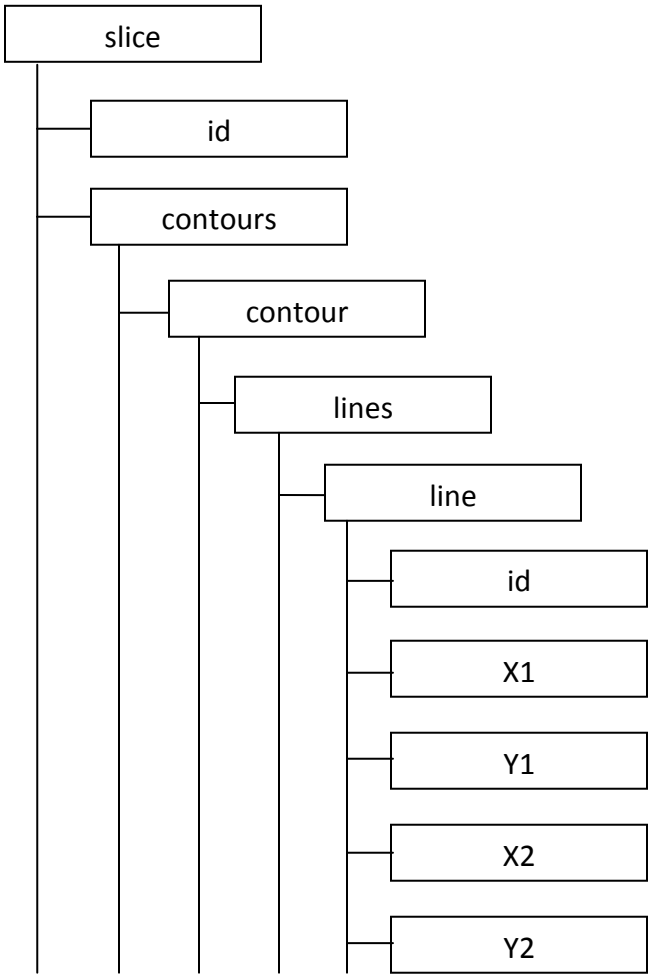


Figure 3-1: Data Structure for slicing output

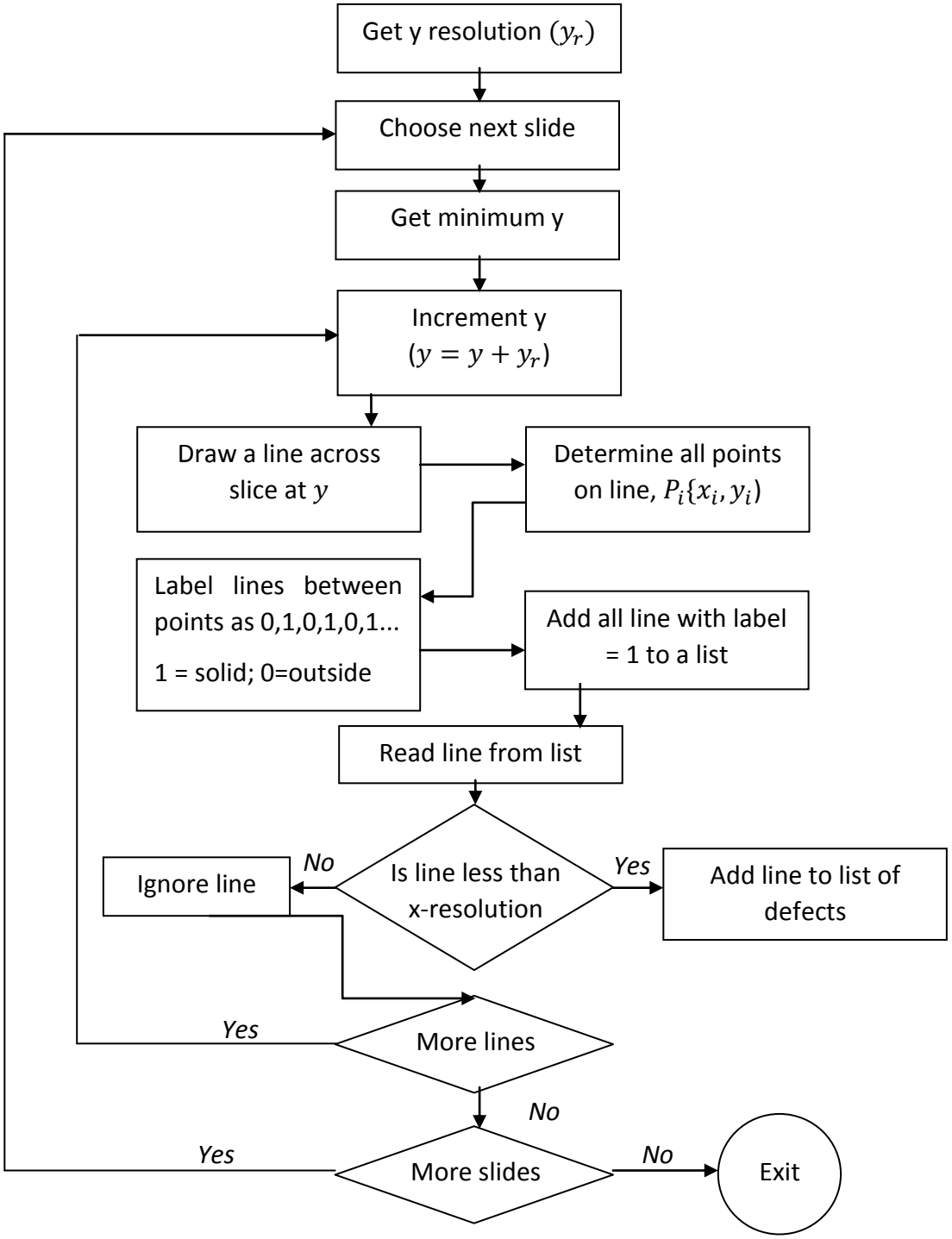


Figure 3-2: Algorithm for determining unprintable regions (defects).

3.3 Ray Casting of 3D slice data

Upon parsing of the 3D slice data for the applicable printer, ray casting is performed on each slice using the resolution in one of x or y direction as the frequency. In Figure 3-2, the algorithm accepts the xyz resolutions quoted by the printer as inputs. The z resolution is taken as the slicing thickness. Arbitrarily y or x is chosen as the direction to begin casting.

For each slice, a line is drawn from one edge of the slice to the other at the y axis. The pitch of this line is equal to the y resolution. All the points where this line intersects with the edge of the contours in the slide is identified and labeled **0** or **1** in alternating order. Lines labeled **1** are for solid regions. Lines labeled **0** are outside the solid regions in the slide; hence they will be used for analysing gaps**.**

When permitted by availability of computing resources, the resolutions can also be adjusted to apply the adaptive slicing methods to address situations where a feature may fall in between slices. Another way to address this will be to add the ray angle parameter during tracing. If a feature/region falls between two lines, increasing the resolution will introduce an extra line that can cut through the feature. Hence increasing the resolution, while adding more computing load to the system, will increase the chances of detecting more features.

3.3.1 SVG Slice input sets

Slic3r is used to generate scalable vector graphics (SVG) slice outputs. This output is rendered as a group of contours enclosed in SVG group (g) element. This output can even be visualized by any modern web browser since they are capable of native rendering of SVG outputs (**Figure 3-3**).

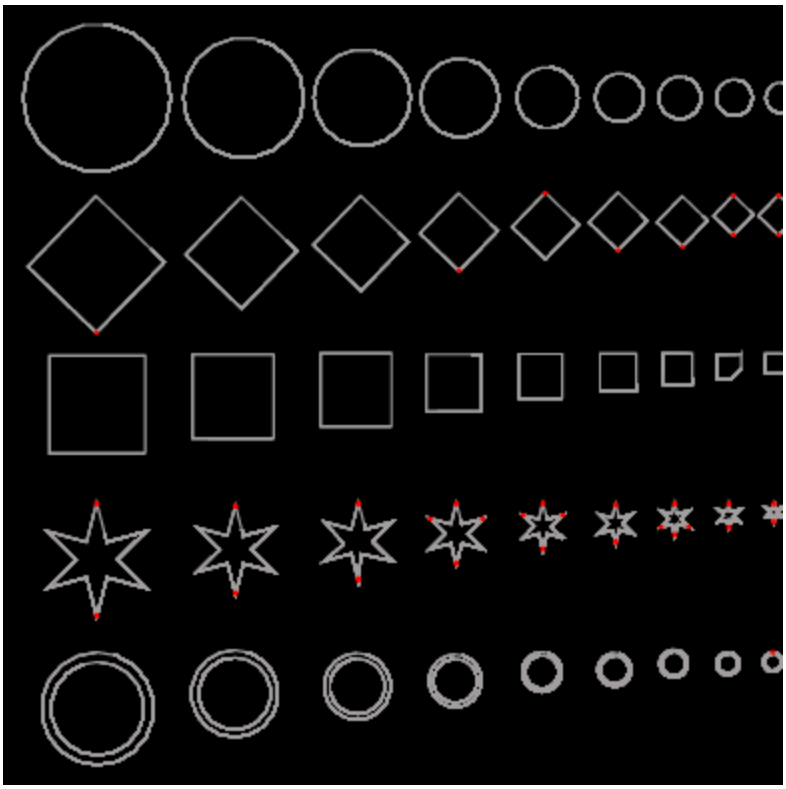


Figure 3-3: Algorithm Browser rendering of SVG data for one slice of regular shapes cross sections.

Using Microsoft Visual Studio 2013[51] .Net tools to convert structured data into object classes, we are able to generate an object class for SVG. This object class is to be used for reading SVG slicing outputs into a list of group elements. A group element then contains a list of points having x and y coordinates. These points define the contours and are referred to as polygons in SVG terms (See **Figure 3-4**).

```
<g xmlns="http://www.w3.org/2000/svg" id="layer0" xmlns:slic3r="http://slic3r.org/namespaces/slic3r" slic3r:z="2.5e-007">

<polygon style="fill: black;" points="27.8733,53.1758 27.2373,53.288 26.6781,53.6109 26.54,53.6905 26.4376,53.8126 26.0225,54.3073 25.8016,54.9142 25.7471,55.064 25.7471,55.8691 25.968,56.476 26.0225,56.6258 26.1249,56.7479 26.54,57.2426 27.0993,57.5655 27.2373,57.6451 27.8733,57.7573 28.0303,57.785 28.6663,57.6728 28.8232,57.6451 28.9613,57.5655 29.5205,57.2426 29.9356,56.7479 30.0381,56.6258 30.0807,56.5088 30.3135,55.8691 30.3135,55.064 30.259,54.9142 30.0381,54.3073 29.623,53.8126 29.5205,53.6905 29.3825,53.6109 28.8232,53.288 28.1872,53.1758 28.0303,53.1481" slic3r:type="hole" />

<polygon style="fill: black;" points="5.1407,53.6138 4.91714,53.6532 3.97362,54.198 3.74076,54.3324
```

```
3.04045,55.167 2.86762,55.373 2.495,56.3968 2.40303,56.6494 2.40303,58.0078 2.77566,59.0316 2.86762,59.2842
3.56793,60.1188 3.74076,60.3248 4.68428,60.8696 4.91714,61.004 5.99007,61.1932 6.25486,61.2399
7.32779,61.0507 7.59259,61.004 8.53611,60.4593 8.76897,60.3248 9.46927,59.4902 9.64211,59.2842
10.0147,58.2605 10.1067,58.0078 10.1067,56.6494 9.73407,55.6257 9.64211,55.373 8.9418,54.5384
8.76897,54.3324 7.82545,53.7877 7.59259,53.6532 6.51966,53.4641 6.25486,53.4174" slic3r:type="hole" />
        </g>
```

Figure 3-4: **SVG <g> output (top) for two circles (bottom)**

Our web application, **SlicingResValidator**, is written in Microsoft C# .Net programming language. It uses the SVG slicing set from Slic3r or other applications as input and a binary or ASCII STL version of the object to be validated.

3.3.2 Pseudo-code/algorithm for line intersection

The algorithm/pseudo-code for determining the intersection points is given Figure 3-5:

```
public PointF FindLineIntersection(PointF start1, PointF end1, PointF start2, PointF end2)
    {
        float denom = ((end1.X - start1.X) * (end2.Y - start2.Y)) - ((end1.Y - start1.Y) * (end2.X - start2.X));
        //  AB & CD are parallel
        if (denom == 0)
            return PointF.Empty;
        float numer = ((start1.Y - start2.Y) * (end2.X - start2.X)) - ((start1.X - start2.X) * (end2.Y - start2.Y));
        float r = numer / denom;
        float numer2 = ((start1.Y - start2.Y) * (end1.X - start1.X)) - ((start1.X - start2.X) * (end1.Y - start1.Y));
        float s = numer2 / denom;
        if ((r < 0 || r > 1) || (s < 0 || s > 1))
            return PointF.Empty;
        // Find intersection point
```

```
        PointF result = new PointF();

        result.X = start1.X + (r * (end1.X - start1.X));

        result.Y = start1.Y + (r * (end1.Y - start1.Y));

        return result;

    }
```

Figure 3-5: SlicingResValidator Intersection Point Detector Algorithm

Our code checks for the intersection of two lines. If an intersection is found, the code also checks if the point of intersection actually occurs on the applicable lines. If the point does not occur on both lines, the algorithm discards the point.

Our application traverses the entire slicing data set to identify defective regions. If the algorithm in SlicingResValidator is applied to slicing outputs, it will detect defective regions as those having sizes/lengths less than the resolution of the applicable printers. Other uses of the algorithm in SlicingResValidator will be when the check is introduced during model design to enforce the requirement to ensure that features/regions do not fall below the printer resolution.

3.3.3 SlicingResValidator, User Interface and Workflow

SlicingResValidator is built on .Net platform with MVC web application framework using C#. The following components are integrated into SlicingResValidator:

- QuantumConcepts STL Parser: This is a proprietary component for reading STL ASCII and binary files. The data is parsed into an object whose class can be used in C# to manipulate and work with the STL dataset
- Three.js: This is a light 3D rendering javascript component.
- Thingiview.js: Makerbot's Thinkiverse 3D viewer implemented in javascript for web rendering of 3D files (STL).

To use SlicingResValidator, the user is required to create a folder for each model to be investigated inside the /Files folder. An STL file of the model named STL.stl is to be included in model folder. This STL file is to be sliced into SVG with Slic3r. The use of Slic3r is optional but is the only validated slicing outputs. Slicer also packages each layer as a group (<g>) in the root node of the SVG. This makes it easy for SlicingResValidator to traverse each slice.

SlicingResValidator accepts the following inputs:

- Input File (3D CAD Model)
- X resolution
- Y resolution

The Z resolution is provided to Slic3r during slicing configuration for printer G-Code generation. SLic3r optionally slices to SVG or machine G-Code. Note that the lower the Z resolution value, the more the number of layers (<g>) generated by Slic3r and the larger the file size.

SlicingResValidator obviously has limitations that depend on the size of the 3D CAD model. As a web based application, there are limitations imposed by serialization. This limitation can be overcome if the application is a desktop forms control or native application. Also, another limitation applies when resolution is increased. If the z or y resolution value is reduced from 1mm to 0.1 mm, this will result in a 10 fold increase in the number of layers, hence increases in file size and computing load.

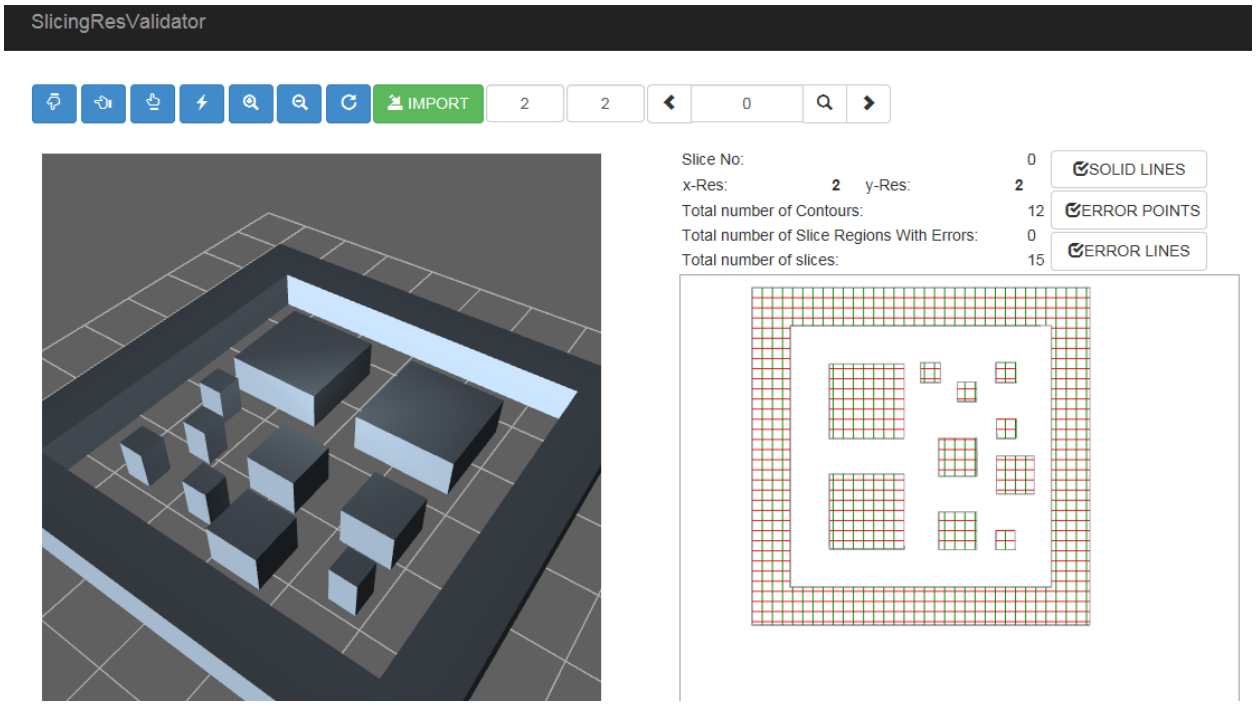


Figure 3-6 SlicingResValidator User Interface

Chapter 4. Results and Validation

Our application, SlicingResValidator accepts a CAD model, its SVG slicing dataset, x resolution and y resolution as inputs. A few 3D CAD models were tested and SlicingResValidator was able to detect the potential defects on the slicing sets. These defects are presented as points on both sides of the edges making up the ray solid region of interest. The defect can also be viewed as a line joining these points to show their length as evidence that they are shorter than the printer resolution in that direction. The following metrics were generated from our model:

- **Slicing Number:** This is the index on the slice in the total SVG slicing dataset.
- **Total Number of Contours:** This is the total number of contours found in the selected slice.
- **Total number of Slice Regions with Errors:** When defects are detected, algorithm provides a count of the defects and presents it to the user.
- **Total Number of Slices:** When the 3D model is cut into slices based on the z resolution, this will be the total number of slices covering the entire 3D model. The number of slices is indirectly proportional to the z value or slicing thickness. The higher the slicing thickness the higher the likelihood of our model not being able to detect defective features/regions. The reason is that features can exist in between the slices and they can be missed.

4.1 Handling of units of measurement

STL format does not require units of measurement to be specified. However, 3D printing is a near net shape process. Hence, it is imperative that the designer measures and constrains the size of the 3D model to match the exact size of the object being replicated. This will also be the case when 3D scanners or medical imaging systems are used to generate the 3D model and its output files. This makes it possible to simply assume that the provided resolution is in a 1:1

scale with the 3D CAD model. Hence, when the height of an object is computed as 10mm, the difference between the maximum z and the minimum z of the CAD file should be exactly equal to 10mm. Therefore, a value of 0.01 for x and 0.01 for y as printer resolutions for test will correspond to 0.01mm for x and 0.01mm for y.

4.2 Validation

To validate our model we created a grid in CorelDraw and placed a model of a slice (layer) of regular squares on the grid. We provided axes information on the grid to indicate size of features and regions. We then exported this model into SVG from CorelDraw and imported this model into SlicingResValidator. We performed the following tests using our validation model

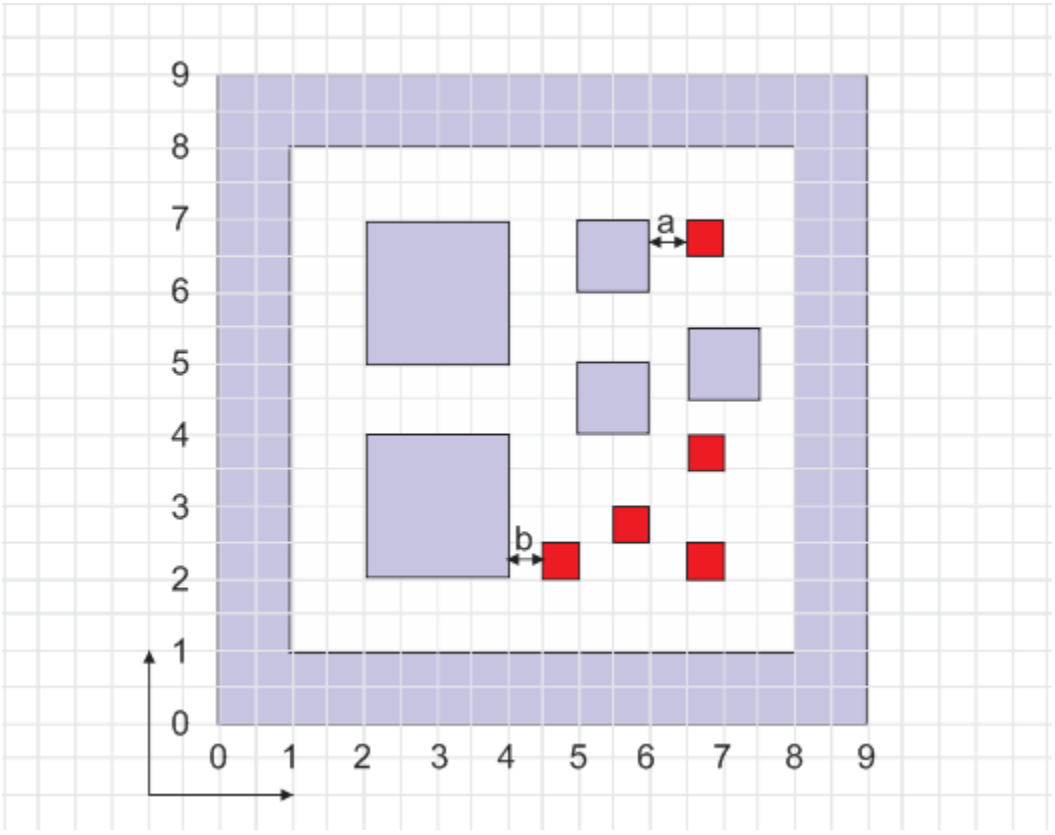


Figure 4-1: 2D Grid Model for Validation

- Check if our method is able to detect the features in red when the resolution value is larger than the size of the features

- Check if our method is able to detect the regions **a** and **b** when the resolution value is larger than the size of the regions
- Check if adjusting the resolution affects the number of errors detected

4.3 Results

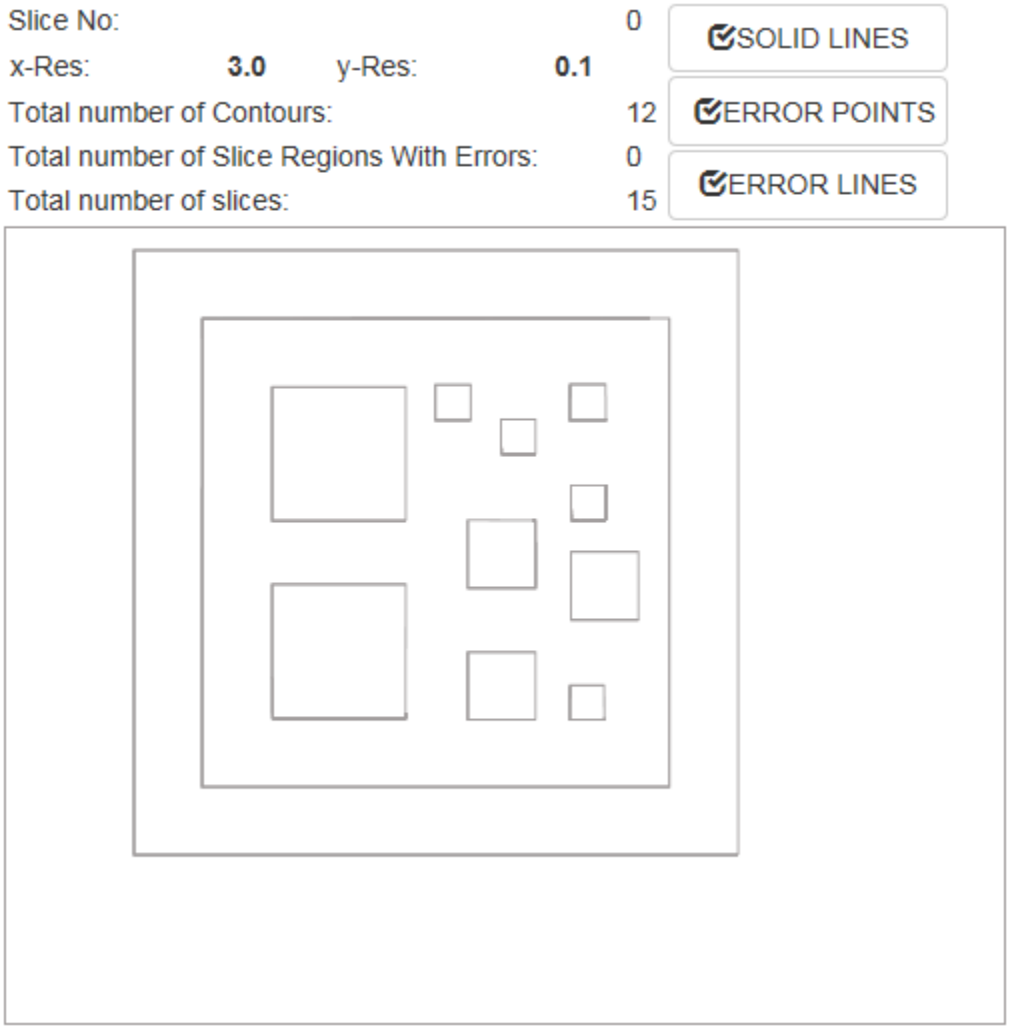


Figure 4-2: SlicingResValidator Output of Validation 2D Model check using x=3.0 and y-0.1 resolutions. The model did not detect any error as expected since the smallest feature is larger than 3.0

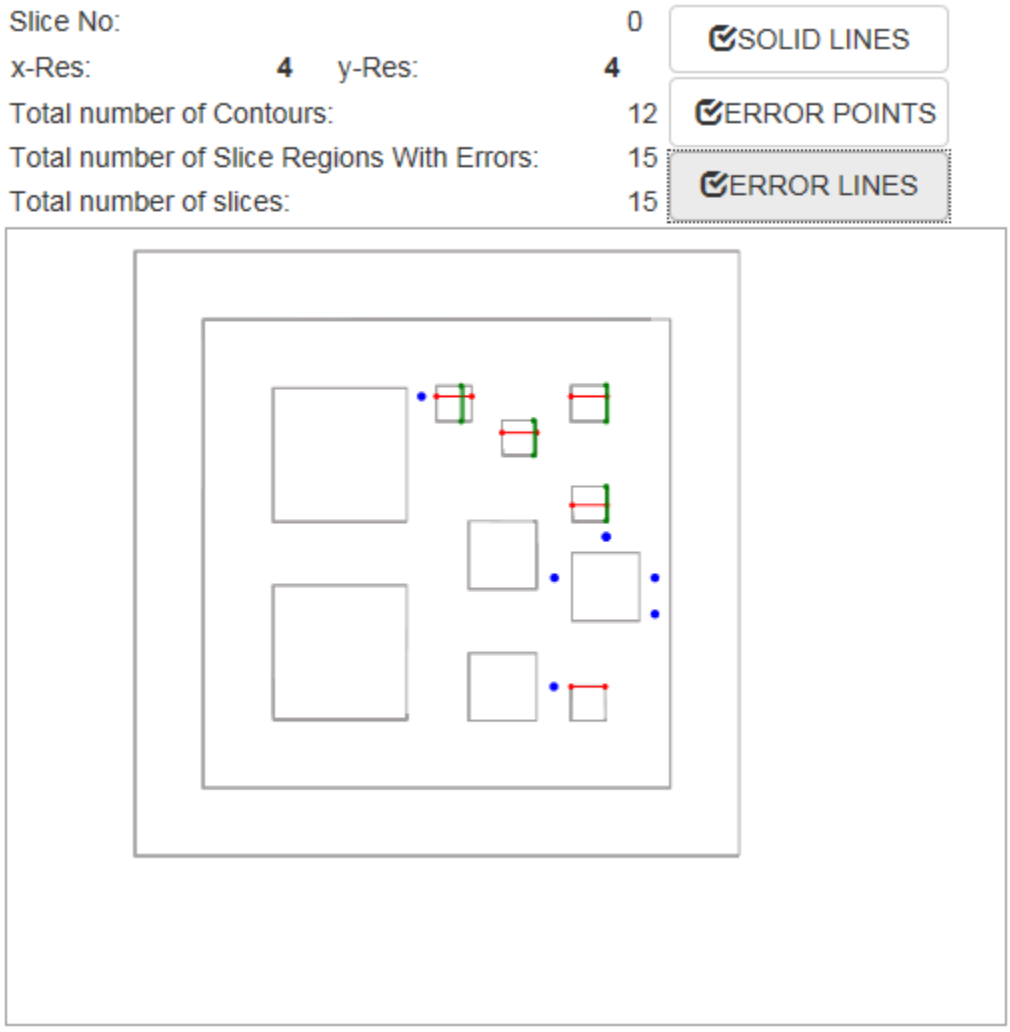


Figure 4-3: SlicingResValidator Output of Validation 2D Model check using x=4.0 and y-4.0 resolutions. Errors were detected in all the squares having sizes less than 4.0. Also errors were detected in gaps with sizes below the resolution values (blue).

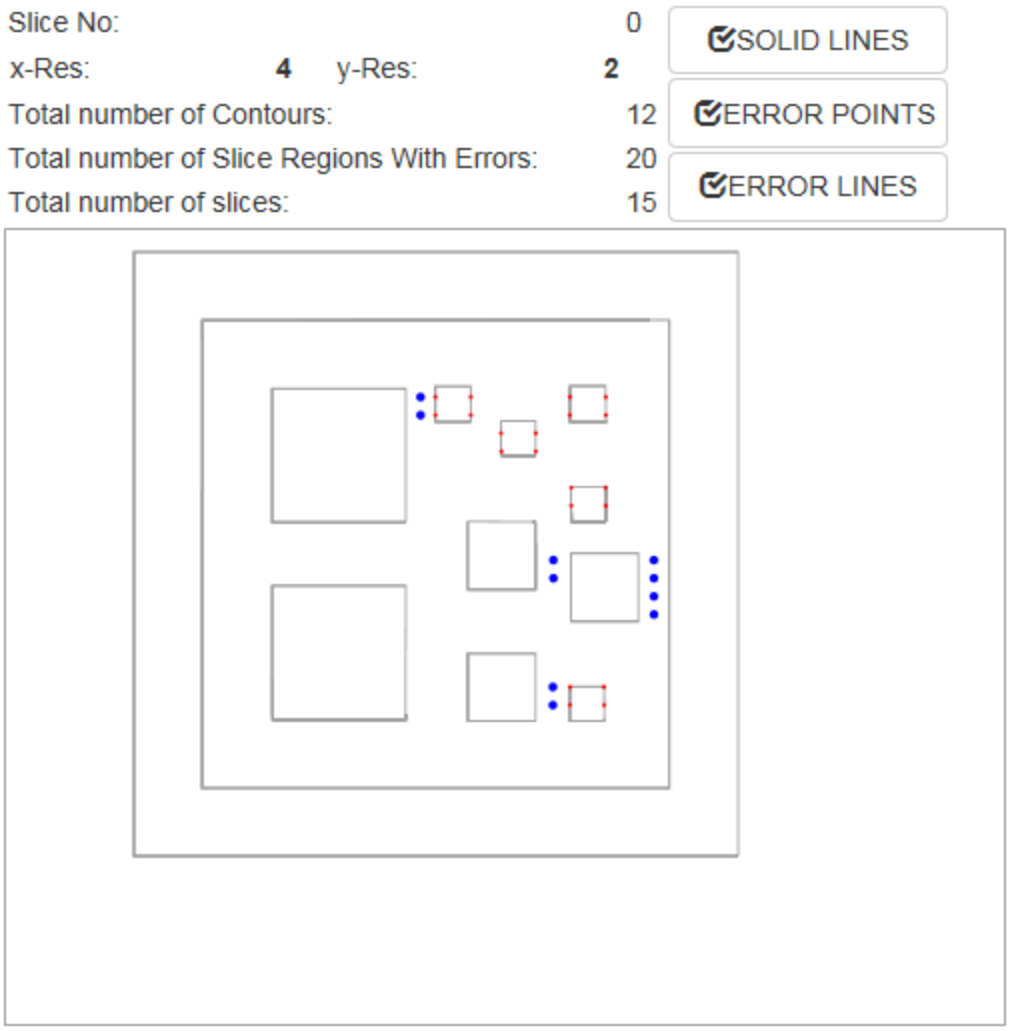


Figure 4-4: SlicingResValidator Output of Validation 2D Model check using x=4.0 and y-2 resolutions. More Errors were detected in all the squares having sizes less than 4.0. As we reduced the resolution value in y direction by half (y=2), the algorithm detected more errors along the y direction by merely sampling (shooting) more rays.

Our algorithm properly rendered the validation model (**Figure 4-2**). The algorithm did not detect any errors when the resolution values are very small (x=3.0 and y=0.1) **Figure 4-2**. The smallest square is of width 3.5. This is in line with our expectations. The algorithm was able to start detecting errors (15) when we increased the resolution values in **Figure 4-3**. This is indicated as red dots or lines on the edges of regions with defects in the x direction and green

dots or lines in the y direction. Hence, no error was detected. However, when we increased this resolution value to x=4.0, y=4.0 the red squares were detected successfully. Also the gaps with width less than 4.0 were flagged as defective. This is indicated as blue dots in both x and y directions. When we kept the x resolution (x=4) the same and reduced the y resolution by half (y=2) **Figure 4-4**, our algorithm detected more errors (5) in the y direction.

The next model (**Figure 4-5**) contained a flat 3D model of regular geometrical shapes in different sizes. The goal was to evaluate the ability of SlicingResValidator to detect regions of sizes lower than that of the applicable printer resolution. We used the following resolution values for each slice x=1, y=1; x=1, y=1.5; x=1, y=2; x=1, y=2.5; x=3, y=3.5; x=1, y=3.5; x=1, y=4 and plotted their regression in **Figure 4-6** - showing how the algorithm is able to detect more errors as we reduce printer resolution values (higher resolution). Note that increasing printer resolution value actually lowers the printer resolution in that particular direction. Resolution in this context is actually the amount of materials (number of dots) that can be deposited in a given straight line for 1-dimensional resolution (say x); area for 2-dimensional resolution (x, y) or volume for 3-dimensional resolution. The results show that our system picked up more errors as we increased the resolution value in y direction.

In **Figure 4-7**, using the same CAD model as above, we kept the y axis values constant (y=1) and varied the x values x=1, y=1; x=1.5, y=1; x=2, y=1; x=2.5, y=1; x=3, y=1; x=3.5, y=1; x=4, y=1. Our regression plot showed that our system is able to detect more errors as the resolution in the y direction is increased.

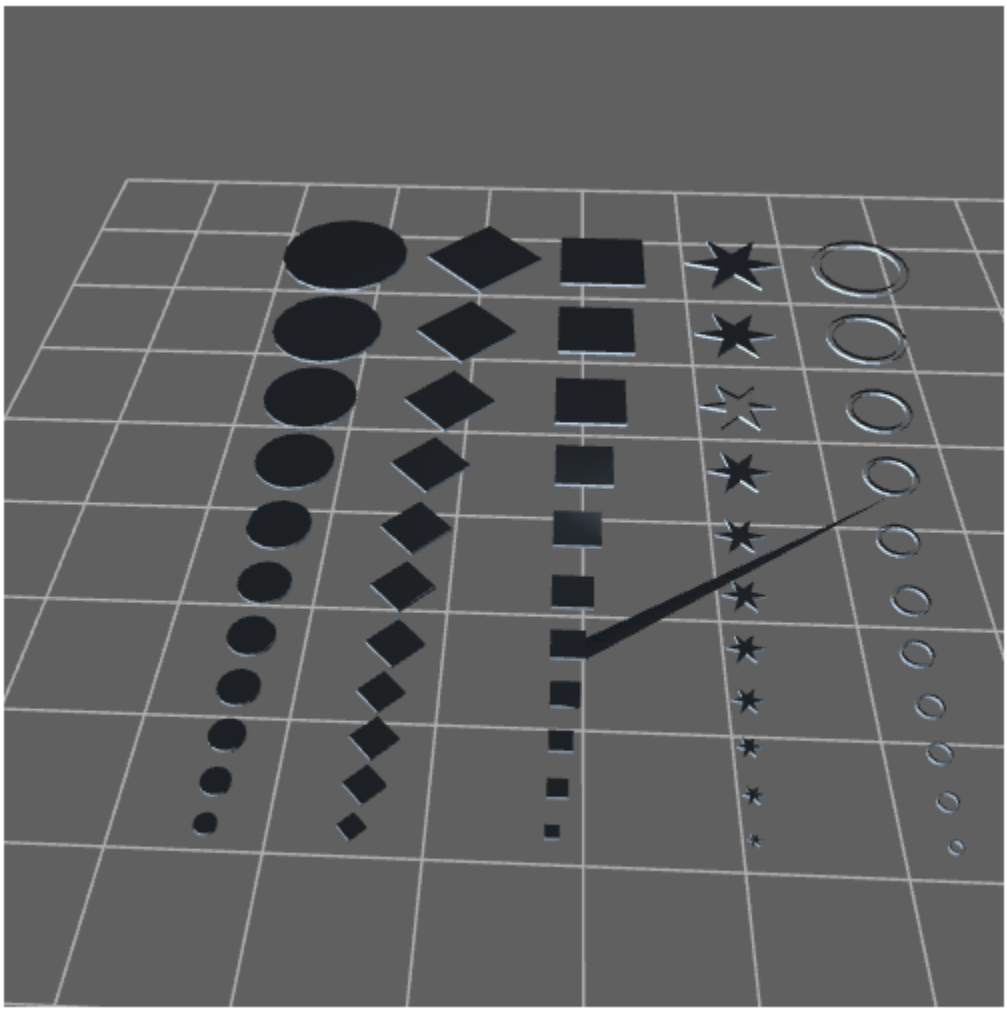


Figure 4-5: 3D CAD Regular Geometrical shapes Checked with SlicingResValidator

Slice No: 0

x-Res: 1 y-Res: 4

Total number of Contours: 65

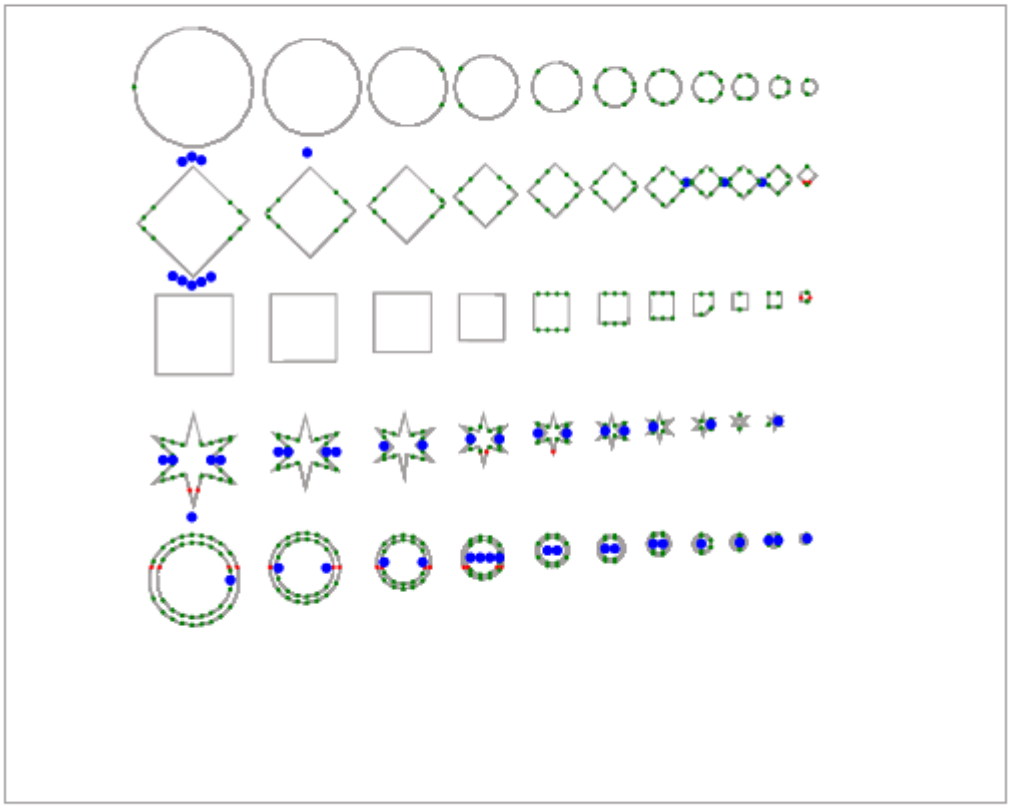
Total number of Slice Regions With Errors: 269

Total number of slices: 1

SOLID LINES

ERROR POINTS

ERROR LINES



x=1

Y	Errors	Regression
1	230	226.2857143
1.5	239	232.4285714
2	238	238.5714286
2.5	226	244.7142857
3	251	250.8571429
3.5	260	257
4	269	263.1428571

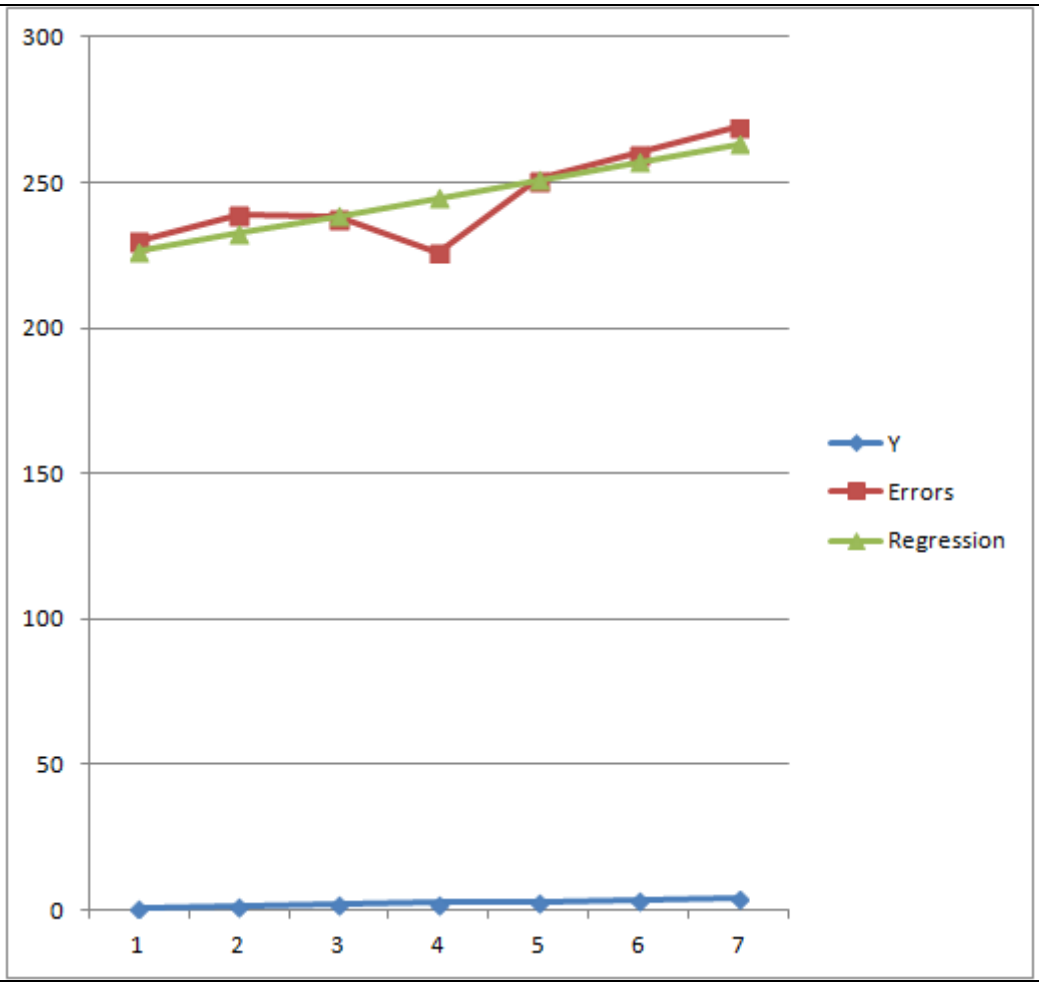


Figure 4-6: Regression plot for SlicingResValidator outputs for a resolution of X=1 and y=1, 1.5, 2, 2.5...

Y=1

X	Errors	Regression
1	230	223.0714286
1.5	248	249.4285714
2	273	275.7857143
2.5	304	302.1428571
3	315	328.5
3.5	353	354.8571429
4	392	381.2142857

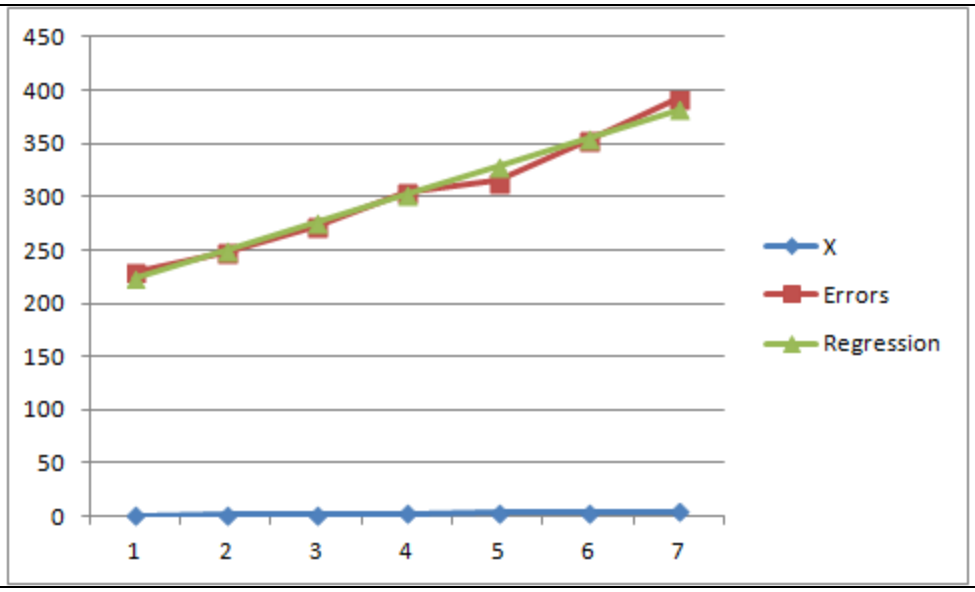


Figure 4-7: Regression plot for SlicingResValidator outputs for a resolution of x=1, 1.5, 2, 2.5... and y=1

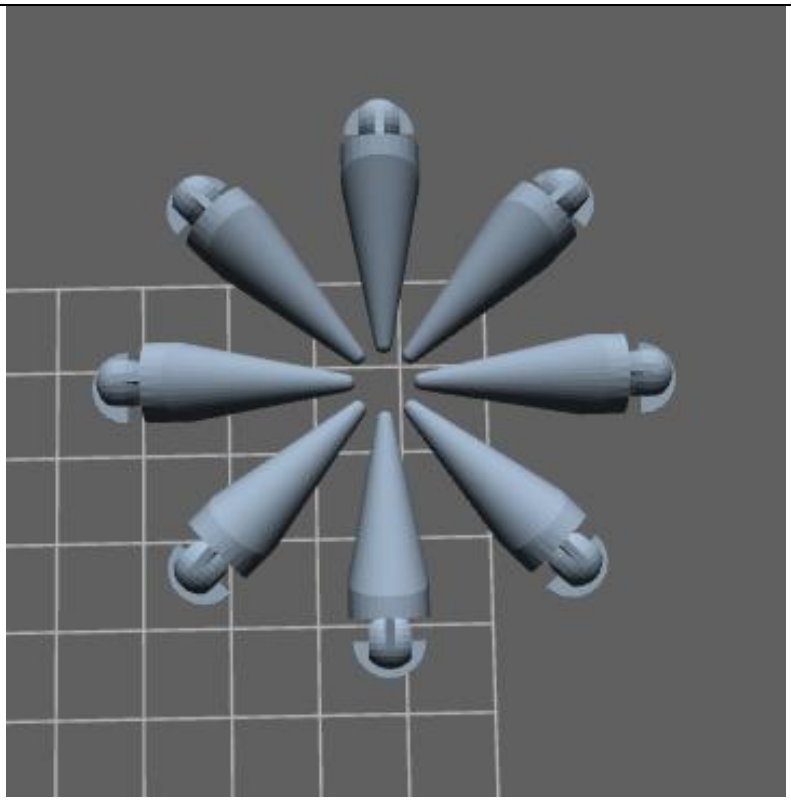


Figure 4-8: 3D CAD Sample Tentacle End Support Checked with SlicingResValidator

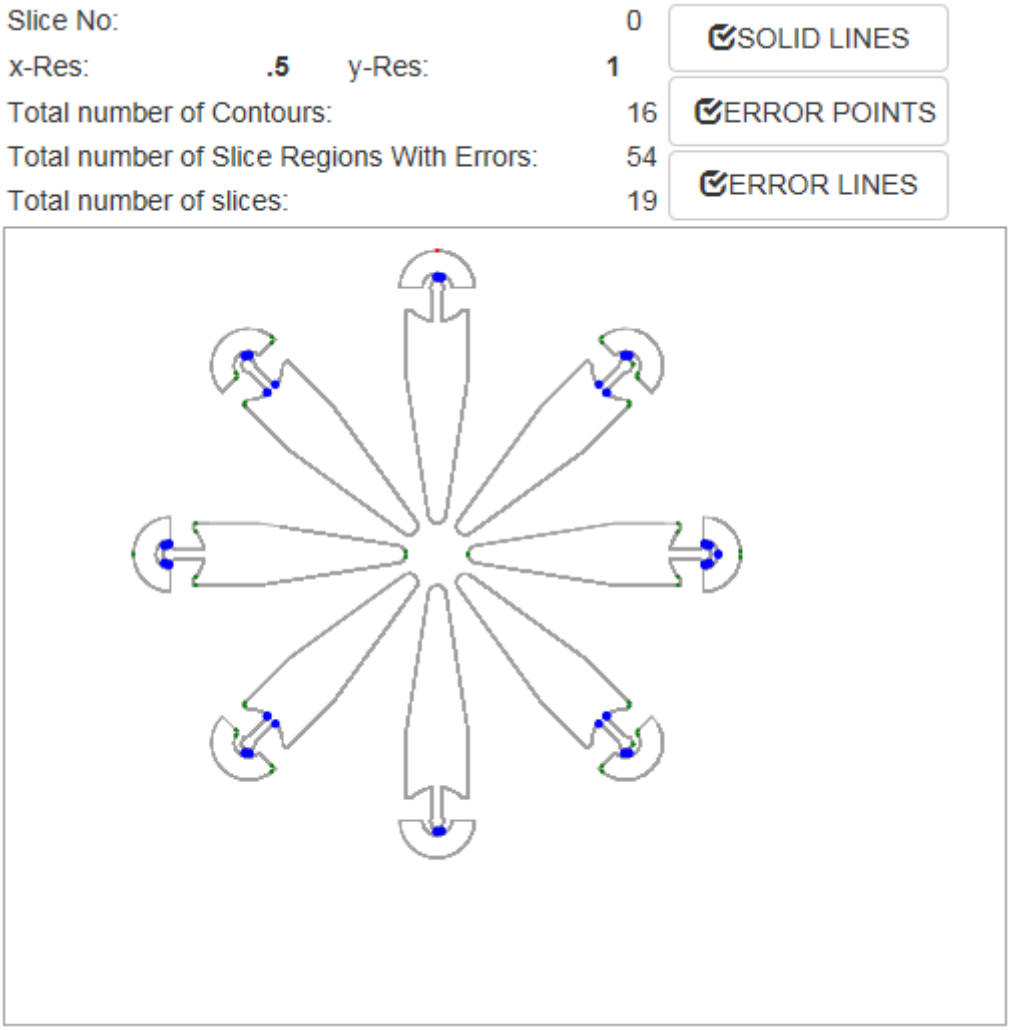


Figure 4-9: Slice number 0 of Sample Tentacle End Support Checked with SlicingResValidator

X=.5		
Y	**Errors**	**Regression**
1	54	59.07142857
1.5	117	111
2	161	162.9285714
2.5	207	214.8571429
3	278	266.7857143
3.5	328	318.7142857
4	359	370.6428571

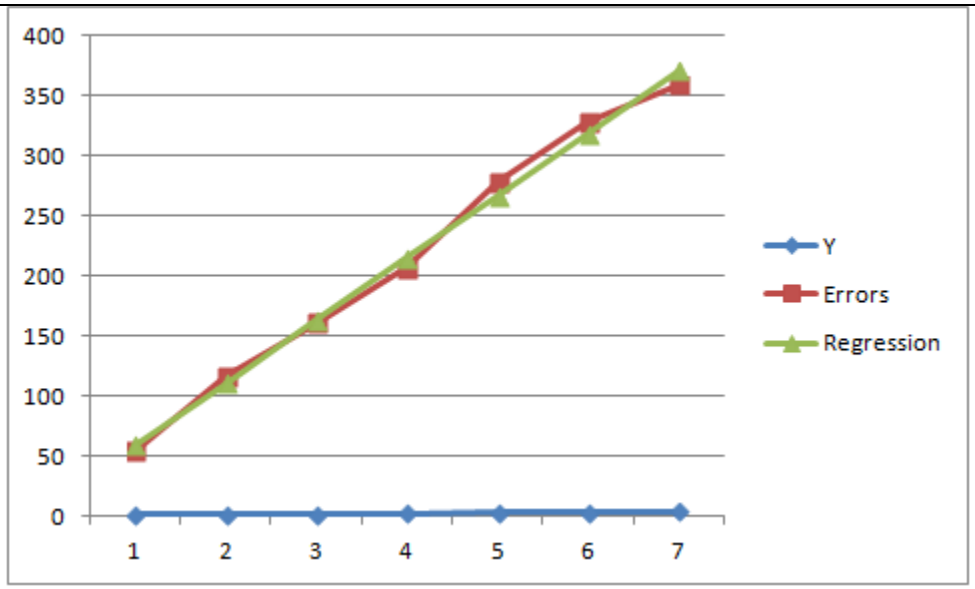


Figure 4-10: Regression plot for SlicingResValidator outputs for Slice number 0 with a resolution of X=0.5 and y=1, 1.5, 2, 2.5...

We also tested our algorithm with more 3D CAD samples from Thingiverse.com. The sample in **Figure 4-8** is called tentacle support. We used Slic3r to slice the sample into 19 SVG <g> sets and rendered it in SlicingResValidator. We then randomly selected slice number 0 **Figure 4-9** and applied the following resolutions settings to test the ability of SlicingResValidator to detect errors in the 3D CAD Sample – x0.5 and y=1.0. Our system was able to detect 54 errors. We then varied the y resolution from 1 to 4 and tested the sample for each resolution keeping x constant (x=1). Our regression plot in **Figure 4-10** showed that our system is able to detect the errors as we increase the resolution. Given that this model is symmetric in several angles, there was actually no utility in changing the x values and keeping the y value constant

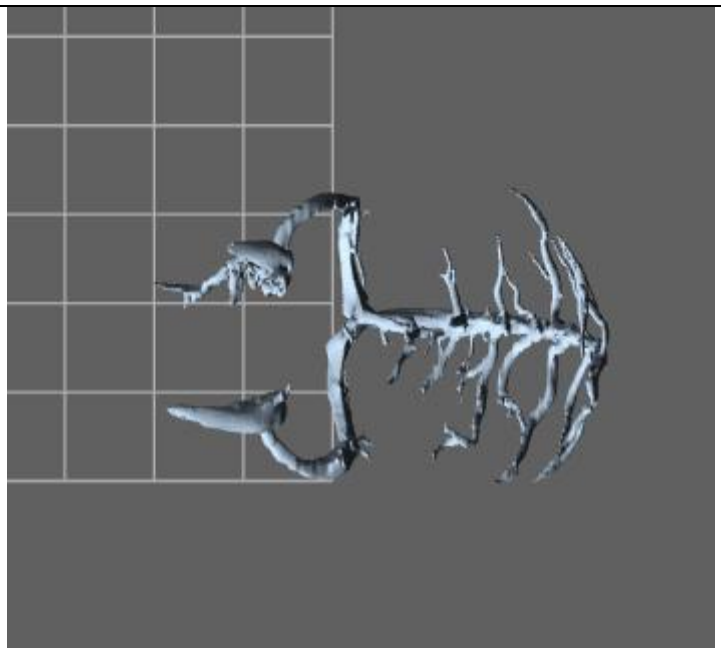


Figure 4-11: 3D CAD Sample of Sinus Veins Checked with SlicingResValidator

As our research also includes investigation into the use of biomedical applications of 3D printing, we also obtained some medical 3D CAD samples from Thingiverse.com and sliced them. The Sinus Vein presented in **Figure 4-11** was sliced with Slic3r and 92 SVG <g> sets were produced to represent the layers. We then randomly selected slice number 30 **Figure 4-12**. With resolution values of x=0.5 and y=1.0, our system detected 24 errors on slice 30 which contained 11 contours (shapes). We then set the x resolution as constant at 0.5 and varied y resolutions (1, 1.5, 2, 2.5, 3, 3.5, 4). The regression plot in **Figure 4-13** showed that our system detected more errors as the resolution value is increased. When we kept the y value constant (at 1) and varied x (1, 1.5, 2, 2.5, 3, 3.5, 4), our system detected more errors as the x values are increased as shown in the regression plot in **Figure 4-15**.

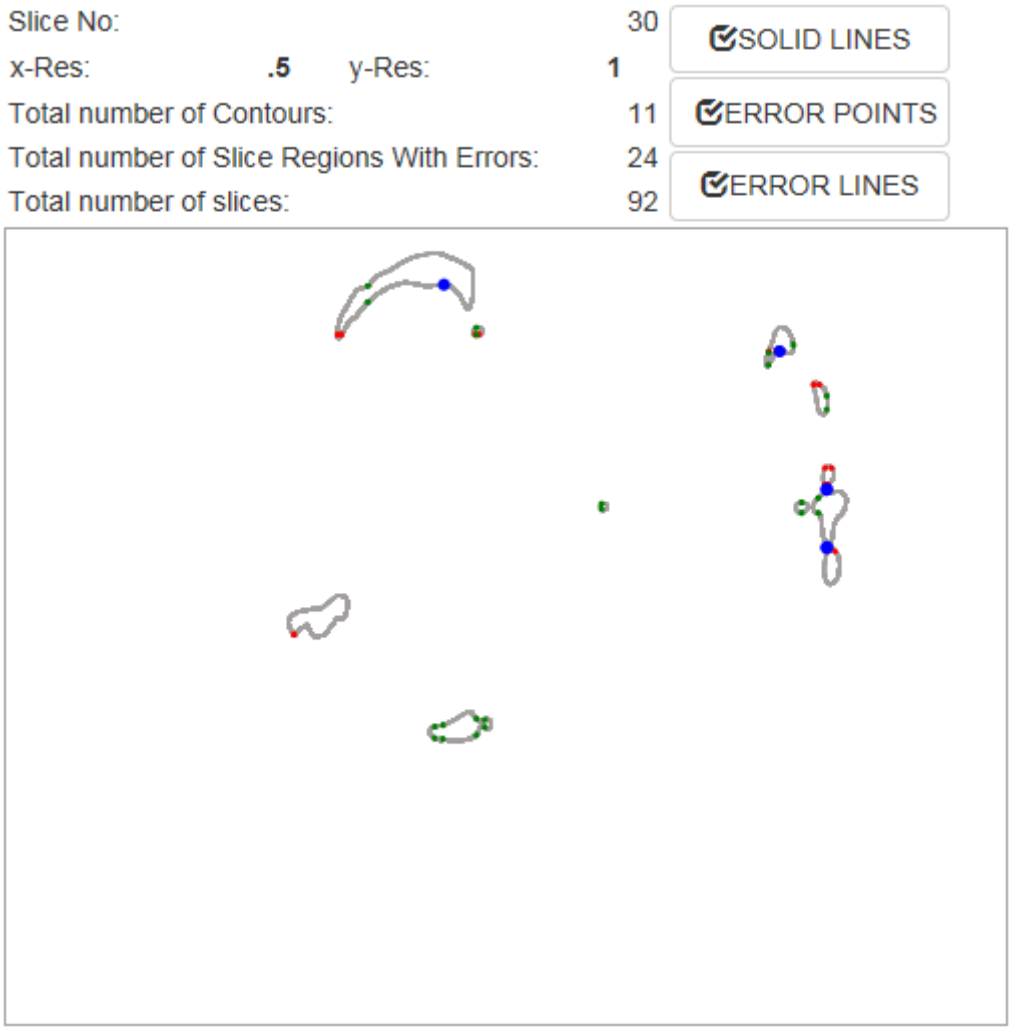


Figure 4-12: Slice number 30 of Sample of Sinus Veins Checked with SlicingResValidator with resolution values x=1, y=0.5

X=.5		
Y	**Errors**	**Regression**
1	40	42.17857143
1.5	40	44.21428571
2	52	46.25
2.5	51	48.28571429
3	52	50.32142857
3.5	52	52.35714286
4	51	54.39285714

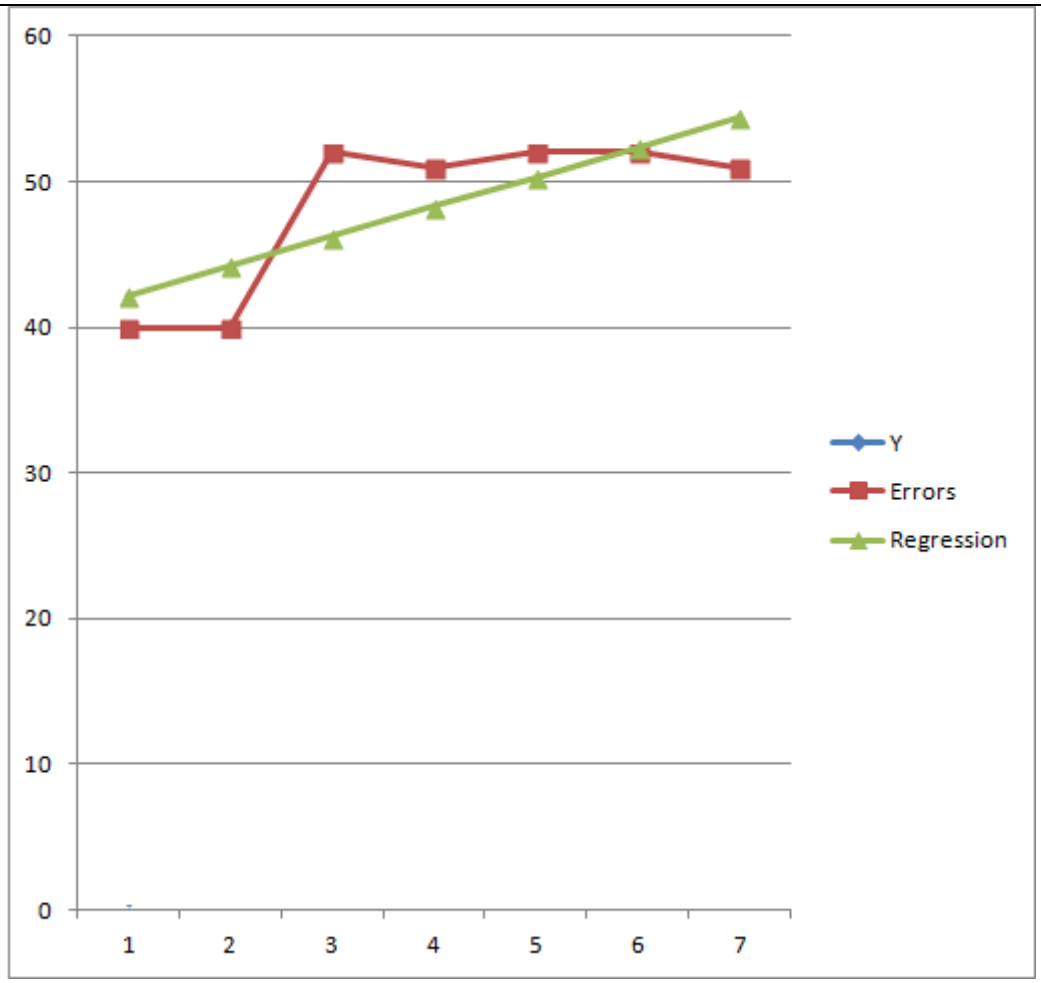


Figure 4-13: Regression plot for SlicingResValidator outputs for Slice number 30 with a resolution of X=0.5 and y=1, 1.5, 2, 2.5...

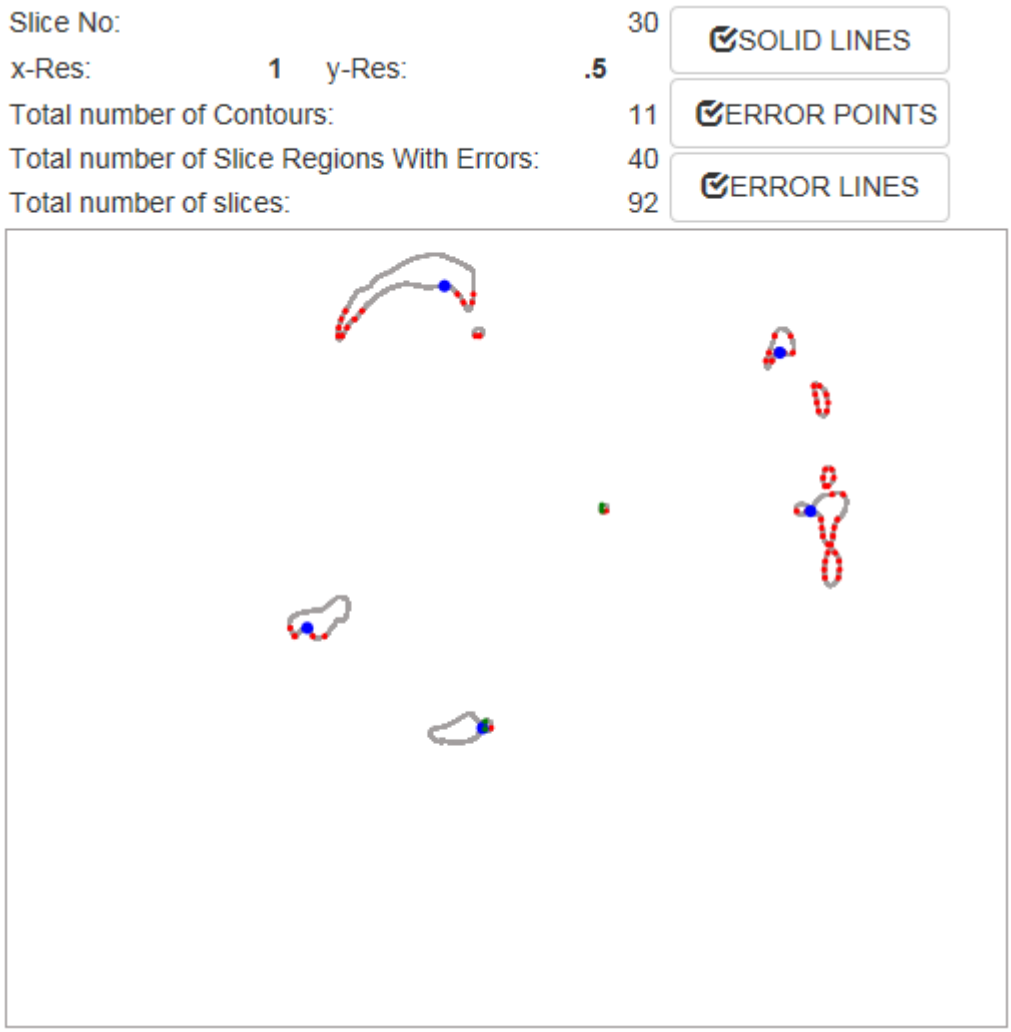


Figure 4-14: Slice number 30 of Sample of Sinus Veins Checked with SlicingResValidator with resolution values x=1, y=0.5

Y=.5		
X	**Errors**	**Regression**
1	40	43.46428571
1.5	47	46.21428571
2	52	48.96428571
2.5	52	51.71428571
3	56	54.46428571
3.5	58	57.21428571
4	57	59.96428571

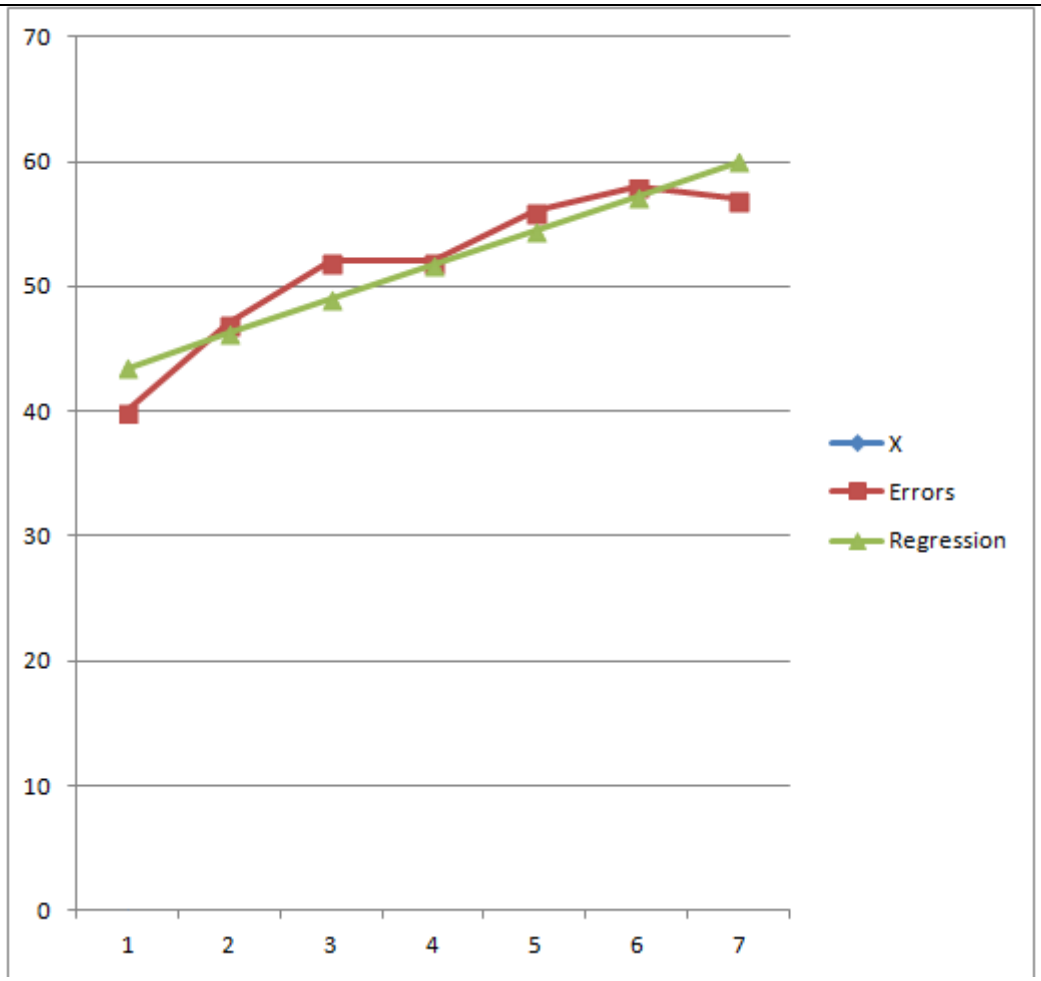
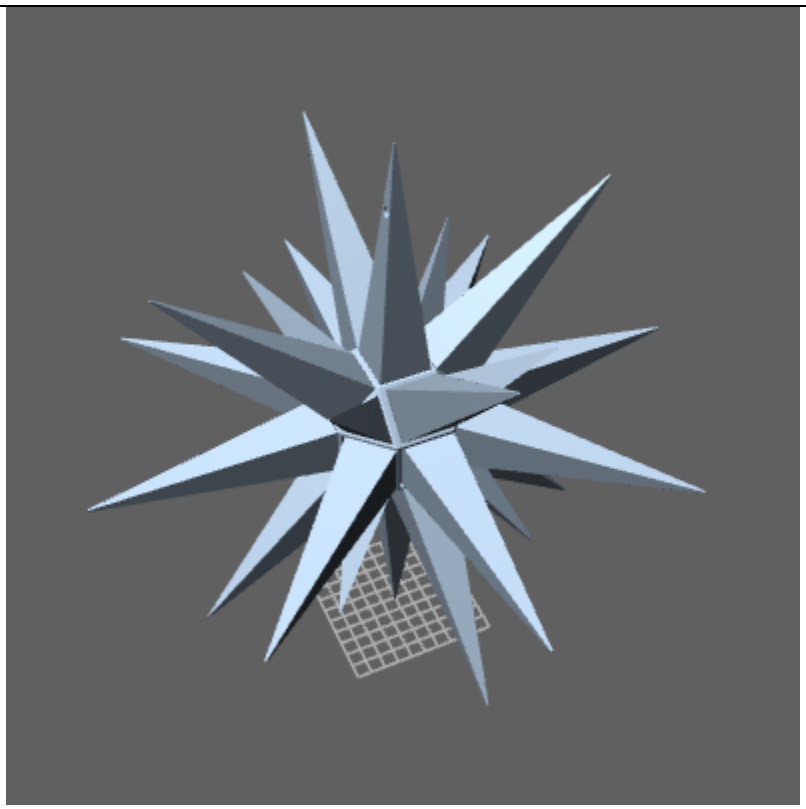
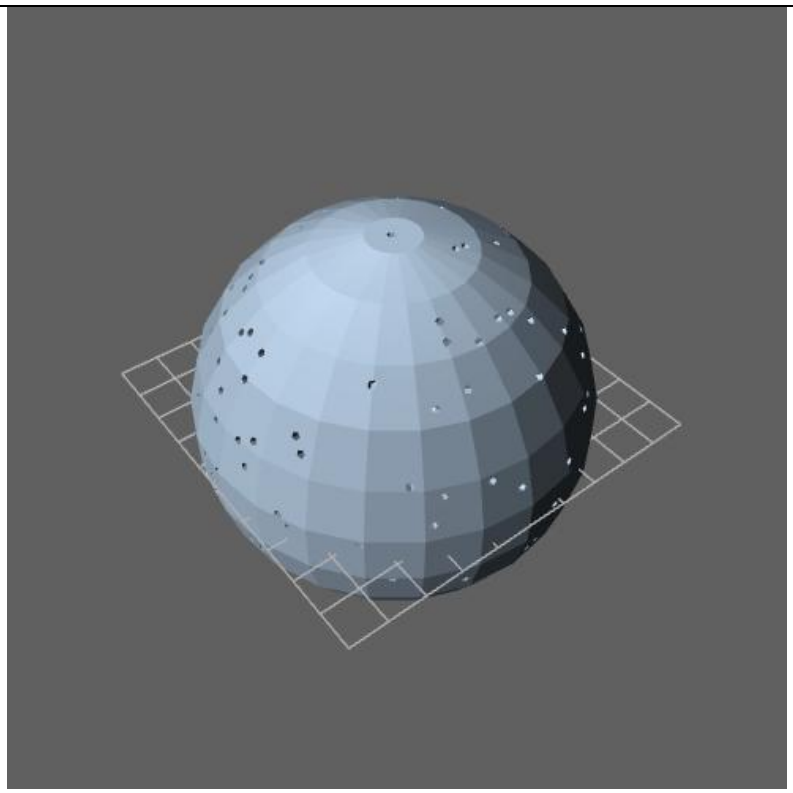


Figure 4-15: Regression plot for SlicingResValidator outputs for Slice number 30 with a resolution of X=1, 1.5, 2, 2.5... and y=1

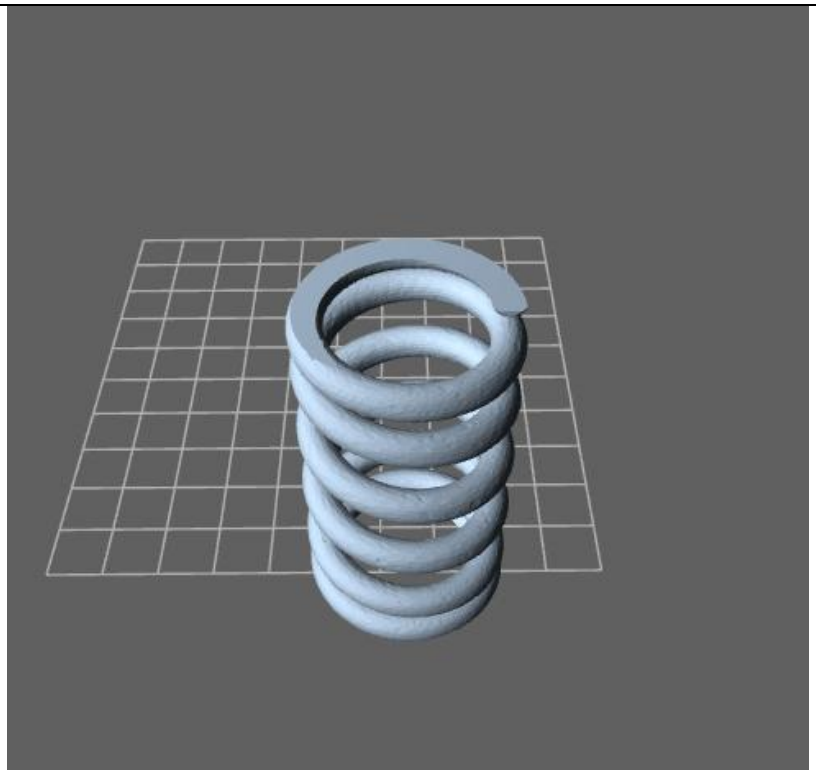
We also tested more 3D samples obtained from Thingivers.com as shown in **Figure 4-15**. In all the samples, our system was able to consistently detect errors as we adjusted the resolution values.



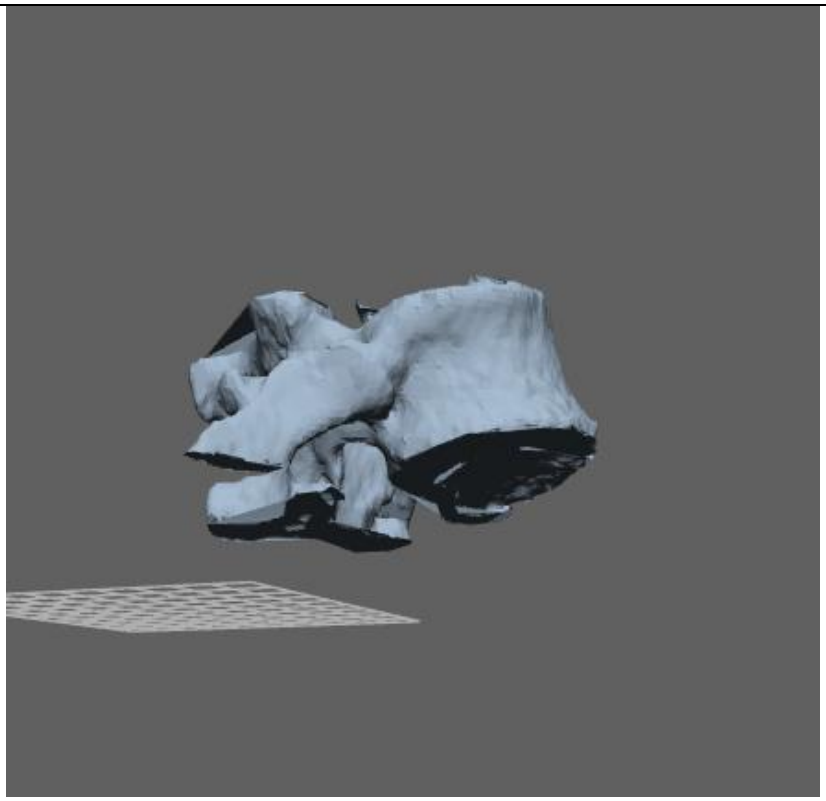
Multistar



CelestialSpherePolar



Spring



Vertebrae

Figure 4-16: More 3D models evaluated by our application

4.4 Discussion

With the increased acceptance of 3D printing, as an alternative to machining in rapid prototyping in professional applications, there is now a need to consider quality assurance of finished products. The quality of 3D printed articles using FDM has been investigated [36] based on the influence of slicing tools on overall quality of 3D printed items. This presented a valid case that the slicing process should be taken into consideration when planning for 3D printing.

We have suggested a novel system for validating the printability of 3D CAD models by a given printer based on the printer resolutions. It provides significant benefits to the quality assurance of 3D printed parts. We showed how our system can utilize the slicing output of 3D CAD data from any slicer application outputting SVG <g> sets and polygons. We demonstrated

that it is possible to visualize, in a slice, regions what will be unprintable in a CAD model, when they fail resolution validations (defective).

The model proposed by Baumann F. et al [36] support our work. It established a framework for testing the quality of slicing outputs of 3D models produced in FDM printers. In their model a few slicing applications were tested and measures were established to evaluate the slicing applications. The resolution of the printer was not included in the framework. We suggested that the validation based on resolution be added to the framework. We proposed a system which can locate the positions where the quality will fail in the actual printing.

An enhancement to our work can be in the area of applying this concept in the 3D model design process. The 3D models come from imaging data that is segmented and consequently stitched up to form a full model. However, other means of generating models include direct design. During model designing, shape operations are very common. They include scaling, stretching or shrinking a model or its features. If the resolution of the printer is added as a constraint, a CAD design application can check if a shape operation is going to result in defective features and warn the user.

Our system did not attempt to propose a correction of CAD models. As an inspection tool, its main focus is to detect and report defects. Also, it is not always practicable to modify 3D models as they may be required to be exact replicas of the physical model. In this case, the utility of our model is in informing the user that the model will have defects when printed by a particular printer with a quoted resolution.

We also acknowledge that, as a study in 3D domain, it would be beneficial to visualize our results in 3D space. We focused on visualizing our results in 2D space since the actual computations (ray casting) are done in 2D space based on slices.

Our system also considered performance in terms of optimal use of computing resources. Depending on the size of the 3D file as well as the complexity of the CAD data, the time it takes our model to scan through a slice and compare/detect regions having defects depends largely on the resolution applied in both directions (x and y). The higher the resolution, the longer it will take to validate the CAD model. Our model did not consider curved shapes, as

is the case with STL. It means that each curve will be represented by discrete lines. Therefore, the number of lines in the polygon, defining each contour in the slides, will influence how long it will take to validate each model or slice. In our model, it took less than one second to process a slice having 195 lines with a resolution of x=0.1 and y=0.1 in the dimensions of the CAD model on a Windows(r) 7 PC with Intel(r) Core i5-4310U CPU @ 2.00GHz 2.60GHz processor running on 8.00 GB ram. It also took less than one second to process this file when the resolution was set at x=0.01 and y=0.01. However, it took approximately 5 seconds to process this same data when the resolution is set to x=0.001 and y=0.001.

Chapter 5. Conclusion

3D printing is an iterative process where the user has to inspect or manually check that features of interest are reproduced as required. When this check produces unsatisfactory result, the CAD data is modified and printed again. For a professional project, this situation might be too expensive. Imperfections can be in the appearance of the printed part or the ability of the printed part to preserve the joints clearances. When the quoted resolution of the applicable printer is larger than the size of the features of interest, these features will not be printed. Instead of waiting for post inspection to make this determination, we have proposed a system to provide this information upfront and save the time and effort and reduce material waste and scrap.

The work by Baumann F. et al [36] proposed a framework for quality assurance of 3D printed items by recognizing that the choice of slicing application influences the outcome of 3D printed parts. We propose that a model for including the printer resolution (x, y and z) be considered in the quality assurance process for 3D printing.

In our solution, we demonstrated that slicing applications can export slice data which is a representation of contours as polygons in 2D space. A slice has the thickness equal to the z resolution. 3D printing as a layered manufacturing process (LM) deposits materials in successive layers to form the complete part. We demonstrated that performing ray casting on slices will provide information regarding the thickness, in either x or y direction, of solid region in a slice. Our model compared the thickness of these regions to the resolution of the printer in that particular direction. Our system was able to identify regions, whose lengths are shorter than the resolution of the printer in that direction, as defective.

We consequently converted our system into an application that can accept 3D STL data and its slicing output in 3D format. This data can be prepared by any slicing application.

As a future work, we proposed that our algorithm be introduced at the design stage. This is to notify 3D model designer, of potential defects when they are about to perform shape

operations that can cause defects. This is when shapes/features/regions will have sizes less than the printer resolution. Other improvements should attempt to use 3D models directly, instead of slices to determine potential defective regions.

We also propose that our algorithm be extended to include the ability to shoot in more angles than just x and y. This will further increase the ability of the tool to detect more errors instead of relying in the increase of number of slices.

References

[1] J. E. Smay, J. Cesarano , J. A. Lewis, Langmuir 2002, 18, 5429

[2] K. Sun, T.-S. Wei, B. Y. Ahn, J. Y. Seo, S. J. Dillon , J. A. Lewis , Adv.Mater. 2013 , 25 , 4539

[3] B. Y. Ahn, E. B. Duoss, M. J. Motala, X. Guo, S. Park , Y. Xiong, J. Yoon, R. G. Nuzzo, J. A. Rogers, J. A. Lewis, Science 2009, 323, 1590.

[4] J. J. Adams, E. B. Duoss, T. F. Malkowski, M. J. Motala, B. Y. Ahn, R. G. Nuzzo, J. T. Bernhard, J. A. Lewis, Adv. Mater. 2011, 23, 1335

[5] Manson P. N., Markowitz B, Mirvis S, DUnham M, Yaremchuk M, Toward CT-Based Facial Fracture Treatment. Plastic Reconstrctiuon Surgery 1990, 85:202-212Heiland M, Schulze D, Rother U, Schmeizle R, In: Lemke HU, Vannier MV, Inamura K, Farman AG, Doi K, Reiber JHC (eds). CARS 2003 – Proceedings of the 17th International Congress and Exhibition: Computer Assisted Radiology and Surgery, 2003. Amsterdam: Elsevier (Excerpta Medica International Congress Series 1256) 2003, pp 1230 – 1234

[6] Khalyfa, A., Vogt, S.; Weisser, J., Grimm, G., Rechtenbach,A., Meyer, W., Schnabelrauch, M. Development of a new calcium phosphate powder-binder system for the 3D printing of patient specific implants, Journal of Materials Science - Materials in Medicine, 18(5), 2007, 909–916.

[7] Stratasys Ltd: Legal Information, 2014, http://www.stratasys.com/legal/legal-information, last accessed 2014-08-27

[8] Deckard, C., Method and apparatus for producing parts by selective sintering, U.S. Patent 4,863,538, filed October 17, 1986, September 5, 1989.

[9] How Direct Metal Laser Sintering Works. THRE3D.com. Retrieved 3 February 2014.

[10] B. Asberg, G. Blanco, P. Bose, J. Garcia-Lopez, M. Overmars, G. Toussaint, G. Wilfong and B. Zhu, Feasibility of design in stereolithography, Algorithmica, Special Issue on Computational Geometry in Manufacturing, Vol. 19, No. 1/2, Sept/Oct, 1997, pp. 61–83.

[11] http://slic3r.org/, G-code generator for 3D printers, last accessed 28-09-2015

[12] http://www.w3.org/TR/SVG/ last accessed 28-09-2015

[13] Munir E. (2013). Slicing 3D CAD Model in STL Format and Laser Path Generation. International Journal of Innovation, Management and Technology, Vol. 4, No. 4, Aug 2013,410-413

[14] http://hotmess3d.com/about-3d-printing, About 3D Printing, last accessed 28-09-2015

[15] http://home.lagoa.com/2014/04/whats-the-right-3d-scanner-for-you/, last accessed 28-09-2015

[16] https://en.wikipedia.org/wiki/Fused_deposition_modeling, Fused Deposition Modeling, last accessed 06-01-2016

[17] http://www.custompartnet.com/wu/fused-deposition-modeling, Fused Deposition Modeling, Last accessed 06-01-2016

[18] Jones, R., Haufe, P., Sells, E., Iravani, P., Olliver, V., Palmer, C., & Bowyer, A. (2011). Reprap-- the replicating rapid prototyper. Robotica, 29(1), 177-191.

[19] Karen M. T, Robert A. H. Electron Beam Freeform Fabrication for Cost Effective Near-Net Shape Manufacturing

[20] http://www.custompartnet.com/wu/stereolithography, Stereolithography, last accessed 06-01-2016

[21] http://www.custompartnet.com/wu/additive-fabrication, Additive fabrication last accessed 28-09-2015

[22] Freedman, David H. Layer By Layer. Technology Review 115.1 (2012): 50–53. Academic Search Premier. Web. 26 July 2013.

[23] http://www.custompartnet.com/wu/selective-laser-sintering, Selective Laser Sintering, last accessed 06-01-2016

[24] StereoLithography Interface Specification, 3D Systems, Inc., July 1988

[25] StereoLithography Interface Specification, 3D Systems, Inc., October 1989

[26] http://www.custompartnet.com/wu/direct-metal-laser-sintering, Direct Metal Laser Sintering, Last accessed 06-01-2016

[27] https://en.wikipedia.org/wiki/STL_(file_format), STL File format. Last accessed 06-01-2015

[28] SLC File Specification, 3D Systems, Inc., 1994

[29] Grimm, Todd (2004), User's Guide to Rapid Prototyping, Society of Manufacturing Engineers, p. 55, ISBN 0-87263-697-6.

[30] ISO 10303-21:2002 Industrial automation systems and integration -- Product data representation and exchange - Part 21: Implementation methods: Clear text encoding of the exchange structure

[31] http://www.fabbers.com/tech/STL_Format last accessed 28-09-2015

[32] Chua, C. K; Leong, K. F.; Lim, C. S. (2003), Rapid Prototyping: Principles and Applications (2nd ed.), World Scientific Publishing Co, ISBN 981-238-117-1 Chapter 6, Rapid Prototyping Formats. Page 237

[33] Topcu, O., Tascioglu, Y., & Unver, H. O. (2011). A Method for Slicing CAD Models in Binary STL Format. 6th International Advanced Technologies Symposium (IATS'11),May 2011, 163, 141-145

[34] EIA Standard RS-274-D Interchangeable Variable Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines, 2001 Eye Street, NW, Washington, D.C. 20006: Electronic Industries Association, February 1979

[35] Zhang Y. M., Li P., Chen Y., Male A. T., Automated system for welding–based rapid prototyping, Mechatronics, 2012,12, 37–53

[36] Baumann F., Buddayci H., Grunert J., Keller F., Roller D., Influence of slicing tools on quality of 3D printed parts, Computer-Aided Design and Applications, August 2015

[37] Dolenc, A. and Makela, I. (1994), "Slicing procedure for layered manufacturing techniques", Computer Aided Design, Vol. 1 No. 2, pp. 4-12.

[38] Sabourin, E., Houser, S.A. and Bohn, J.H. (1996), "Adaptive slicing using stepwise uniform refinementuser, S.A. and Bping Journal, Vol. 2 No. 4, pp. 20-6.

[39] Tyberg, J. and Bohn, J.H. (1998), "Local adaptive slicing", Rapid Prototyping Journal, Vol. 4 No. 3, pp. 118-27.

[40] Sabourin, E., Houser, S.A. and Bohn, J.H. (1997), "Accurate exterior, fast interior layered manufacturing", Rapid Prototyping Journal, Vol. 3 No. 2, pp. 44-52.

[41] Tata, K., Fadel, G., Bagchi, A. and Aziz, N. (1998), "Efficient slicing for layered manufacturing", Rapid Prototyping Journal, Vol. 4 No. 4, pp. 151-67.

[42] Cormier, D., Unnanon, K. and Sanni, E. (2000), “Specifying non-uniform cusp heights as a potential for adaptive slicing”, Rapid Prototyping Journal, Vol. 6 No. 3, pp. 204-11.

[43] Pandey, P.M., Reddy, N.V. and Dhande, S.G. (2003a), “Improvement of surface finish by staircase machining in fused deposition modelling”, Journal of Material Processing Technology, Vol. 132 No. 1, pp. 323-31.

[44] Pulak M. P, Venkata N. R, Sanjay G. D, Slicing Procedures in Layered Manufacturing: a review, Rapid Protoyping Journal, Vol. 9 No.5, June 2003, pp.274-288

[45] Woop, Sven; Schmittler, Jörg; Slusallek, Philipp (2005), "RPU: A Programmable Ray Processing Unit for Realtime Ray Tracing", Siggraph 2005 **24** (3): 434

[46] Daniel Weiskopf (2006). GPU-Based Interactive Visualization Techniques. Springer Science & Business Media. p. 21. ISBN 978-3-540-33263-3.

[47] Roth, Scott D. (February 1982), "Ray Casting for Modeling Solids", Computer Graphics and Image Processing 18 (2): 109–144Ustyantsev, A. gCodeViewer, http://gcode.ws, last accessed 27-08-2015

[48] KISSlicer.com KISSlicer (Keep it Simple Slicer), http://kisslicer.com/, last accessed 27-08-2015Skeinforge Skeinforge – Fabmetheus, http://fabmetheus.crsndoo.com/, last accessed 27-08-2015

[49] Ranellucci, A.: Slic3r G-Code generator for 3D printers, http://slic3r.org/, last accessed August 27 2015

[50] Makerbot Industries LLC: Makerbot Desktop, http://www.makerbot.com/makerware, last accessed 27-08-2015

[51] https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx last accessed 2015-09-19

[52] http://www.thingiverse.com/thing:36413 last accessed 2015-09-19

[53] Richard B., Dominic E., Abby P., The application of Advanced Design and Rapid Prototyping Techniques in Medicine , Medical Modeling (Second Edition), 2015, Pages 99-472

[54] Romano P. 3D Printing Surgical Models of Organs; or Lunch; Phone Screens; How To Do It (3D) Yourself; Computers that Track Your Eyes and Take Eye Commands. Binocul Vis Strabolog Q Simms Romano 2013;28:121–128.

[55] Michalski M, Ross J, The shape of things to come: 3D printing in medicine. JAMA 2014;312:2213–2214.

[56] Ventola C., Medical applications for 3D Printing: current and projected uses. P T 2014;39:704–711.

[57] Biron VL, Gross M, Broad R, et al. Radial forearm free flap with titanium mesh sandwich reconstruction in complex anterior skull base defects. J Craniofac Surg 2012;23:1763–1765.

[58] Lethaus B, Kessler P, Boeckman R, et al. Reconstruction of a maxillary defect with a fibula graft and titanium mesh using CAD/CAM techniques. Head Face Med 2010;6:16.

[59] Zinser M., Mischkowski R., Sailer H., et al. Computer-assisted orthognathic surgery: feasibility study using multiple CAD/CAM surgical splints. Oral Surg Oral Med Oral Pathol Oral Radiol 2012;113:673–687.

[60] Okumura H, Chen LH, Yokoe Y, et al. CAD/CAM fabrication of occlusal splints for orthognathic surgery. J Clin Orthod 1999;33:231–235.

[61] Xu X, Ping FY, Chen J, et al. Application of CAD/CAM techniques in mandible large-scale defect and reconstruction with vascularized fibular bone graft. Zhejiang Da Xue Xue Bao Yi Xue Ban 2007;36:498–502.

[62] Jonathan S, Braden C, Vincent A, Anees F,M, Changyong F, Sarah P, Jean J,Ahmed G, A Novel technique for Simulated Surgical Procedure using 3D printing technology, Journal of Urology Vol. 193, No. 4S, Supplement, May 16, 2015

[63] http://www.cartridgesave.co.uk/news/3d-printing-for-medical-research/ last accessed 22-09-2015

[64] http://www.wired.com/2013/07/is-this-cast-the-future-of-healing-broken-bones/ last accessed 22-09-2015

[65] http://www.3d-printing.net/content/another-3d-printing-medical-breakthrough-3d-printed-face last accessed 22-09-2015

[66] http://fredlybrand.com/2013/06/28/3d-printing-chasm-crossing-applications/ last accessed 22-09-2015

[67] http://www.abc.net.au/news/2015-06-20/melbourne-man-receives-titanium-3d-printed-prosthetic-jaw/6536788 last accessed 28-09-2015

[68] http://reprap.org/wiki/G-code last visited 06-01-2016

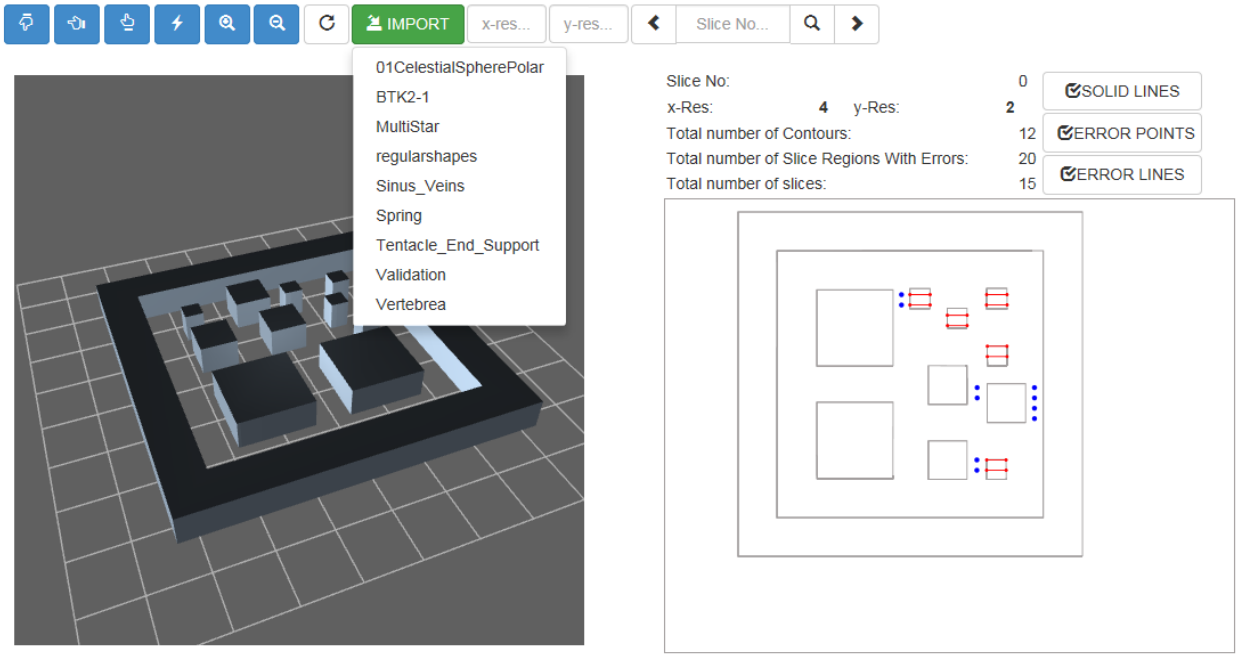
[69] http://www.seagullscientific.com/label-software/barcode-label-design-and-printing/?gclid=CIfGrM6mlsoCFQaraQoddG4GDA last accessed 06-01-2016

[70] Ray-tracing and other Rendering Approaches, lecture notes, MSc Computer Animation and Visual Effects, Jon Macey, University of Bournemouth

[71] Weisstein, Eric W. Line-Line Intersection. http://mathworld.wolfram.com/Line-LineIntersection.html last accessed

.

Appendix A: User Interface



Appendix B: Source Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml.Serialization;
namespace SlicingResValidator.Models
{
    [Serializable]
    public class slice2
    {
        [XmlAttribute("id")]
        public string id { get; set; }
        [XmlAttribute("total")]
        public int total { get; set; }
        [XmlAttribute("totalregionwitherrors")]
        public int totalregionwitherrors { get; set; }
        [XmlAttribute("width")]
        public double width { get; set; }
        [XmlAttribute("height")]
        public double height { get; set; }
        [XmlAttribute("minx")]
        public double minx { get; set; }
        [XmlAttribute("miny")]
        public double miny { get; set; }

        [XmlAttribute("maxx")]
        public double maxx { get; set; }
        [XmlAttribute("maxy")]
        public double maxy { get; set; }
        public List<contour> contours { get; set; }
        public List<cline> xerrorlines { get; set; }
        public List<List<ipoint>> xintersectionpoints { get; set; }
        public List<List<cline>> xintersectionlines { get; set; }
        public List<List<cline>> xsolidlines { get; set; }
        public List<List<ipoint>> xsolidpoints { get; set; }
        public List<List<cline>> xreserrorlines { get; set; }
```

```
    public List<List<cline>> ixreserrorlines { get; set; }

    public List<cline> yerrorlines { get; set; }
    public List<List<ipoint>> yintersectionpoints { get; set; }
    public List<List<cline>> yintersectionlines { get; set; }
    public List<List<cline>> ysolidlines { get; set; }
    public List<List<ipoint>> ysolidpoints { get; set; }
    public List<List<cline>> yreserrorlines { get; set; }
    public List<List<cline>> iyreserrorlines { get; set; }
}
[Serializable]
public class STL2
{
    public int fctcnt { get; set; }
    public List<slice> slices { get; set; }
    public Decimal[] vertexlist { get; set; }
    public Decimal[] normallist { get; set; }
    public List<facet> fct { get; set; }
    public List<facet> unusedfct { get; set; }
    public Decimal minX { get; set; }
    public Decimal maxX { get; set; }
    public Decimal minY { get; set; }
    public Decimal maxY { get; set; }
    public Decimal minZ { get; set; }
    public Decimal maxZ { get; set; }
}
[Serializable]
public class contour
{
    private List<cline> ls = new List<cline>();
    public contour()
    {
        this.lines = ls;
    }
    public int id { get; set; }
    public string key { get; set; }
    public List<cline> lines { get; set; }
}
[Serializable]
public class cline
```

```
{
    [XmlAttribute("x1")]
    public double x1 { get; set; }
    [XmlAttribute("y1")]
    public double y1 { get; set; }

    [XmlAttribute("x2")]
    public double x2 { get; set; }
    [XmlAttribute("y2")]
    public double y2 { get; set; }

    //[XmlAttribute("z")]
    //public Decimal z { get; set; }
}
[Serializable]
public class line
{
    [XmlAttribute("f")]
    public int f { get; set; }
    [XmlAttribute("g")]
    public int g { get; set; }
    //[XmlAttribute("id")]
    //public int id { get; set; }
    [XmlAttribute("x1")]
    public Decimal x1 { get; set; }
    [XmlAttribute("y1")]
    public Decimal y1 { get; set; }

    [XmlAttribute("x2")]
    public Decimal x2 { get; set; }
    [XmlAttribute("y2")]
    public Decimal y2 { get; set; }

    //[XmlAttribute("z")]
    //public Decimal z { get; set; }
}
[Serializable]
public class point
{
    [XmlAttribute("f")]
```

```
    public int f { get; set; }
    [XmlAttribute("g")]
    public int g { get; set; }
    [XmlAttribute("x")]
    public Decimal x { get; set; }
    [XmlAttribute("y")]
    public Decimal y { get; set; }

}
[Serializable]
public class ipoint
{
    [XmlAttribute("x")]
    public double x { get; set; }
    [XmlAttribute("y")]
    public double y { get; set; }
    [XmlAttribute("sid")]
    public string sid { get; set; }
    [XmlAttribute("cid")]
    public string cid { get; set; }
}
[Serializable]
public class STL
{
    public Decimal[] vertexlist { get; set; }
    public Decimal[] normallist { get; set; }
    public facet[] fct { get; set; }
    public Decimal minX { get; set; }
    public Decimal maxX { get; set; }
    public Decimal minY { get; set; }
    public Decimal maxY { get; set; }
    public Decimal minZ { get; set; }
    public Decimal maxZ { get; set; }

}
[Serializable]
public class slice
{
    [XmlAttribute("cnt")]
    public int cnt { get; set; }
```

```
    public List<facet> facets { get; set; }
    public List<point> points { get; set; }
    public List<line> lines { get; set; }
    [XmlAttribute("no")]
    public int no { get; set; }
    [XmlAttribute("minZ")]
    public Decimal minZ { get; set; }
    [XmlAttribute("maxZ")]
    public Decimal maxZ { get; set; }
}
[Serializable]
public class facet
{
    [XmlAttribute("grp")]
    public int grp { get; set; }
    [XmlAttribute("id")]
    public int id { get; set; }
    public normal nrm { get; set; }
    public vertex vtx1 { get; set; }
    public vertex vtx2 { get; set; }
    public vertex vtx3 { get; set; }
}
[Serializable]
public class vertex
{
    [XmlAttribute("id")]
    public int id { get; set; }
    [XmlAttribute("x")]
    public Decimal x { get; set; }
    [XmlAttribute("y")]
    public Decimal y { get; set; }
    [XmlAttribute("z")]
    public Decimal z { get; set; }
}
[Serializable]
public class normal
{
    [XmlAttribute("id")]
    public int id { get; set; }
    [XmlAttribute("x")]
```

```
        public Decimal x { get; set; }
        [XmlAttribute("y")]
        public Decimal y { get; set; }
        [XmlAttribute("z")]
        public Decimal z { get; set; }
    }
}

public JsonResult LoadSVG(int i, string xres, string yres, string filename)
        {
            DateTime start = DateTime.Now;
            JsonResult _j = new JsonResult();
            TextReader reader = null;
            slice2 slc = new slice2(); slc.minx = 0; slc.miny = 0; slc.maxx = 0; slc.maxy = 0;
            List<cline> alllines = new List<cline>();
            try
            {

                int h = i;
                XmlSerializer serializer = new XmlSerializer(typeof(svg));
                FileStream fs = new FileStream(Server.MapPath("/Files/" + filename + "/SVG.svg"), FileMode.OpenOrCreate);
                reader = new StreamReader(fs);
                svg svg;
                svg = (svg)serializer.Deserialize(reader);
                reader.Close();

                if (h >= svg.g.Length)
                {
                    _j.Data = "No slice";
                    return _j;
                }
                //for (int h = 0; h < svg.g.Length; h++)
                //{
                slc.id = svg.g[h].id;
                slc.width = (double)svg.width;
                slc.height = (double)svg.height;
                slc.total = svg.g.Length;
```

```
                List<contour> cntrs = new List<contour>();
                bool assigned = false;

                for (int v = 0; v < svg.g[h].polygon.Length; v++)
                {
                    contour cntr = new contour();
                    cntr.id = v;
                    string[]                    fields                    =
svg.g[h].polygon[v].points.Split(' ');
                    List<cline> lines = new List<cline>();
                    for (int k = 0; k < fields.Length - 1; k++)
                    {
                        if (fields[k] != "")
                        {
                            cline ln = new cline();
                            string[] _fields1 = fields[k].Split(',');
                            string[]    _fields2    =    fields[k    +
1].Split(',');
                            ln.x1 = double.Parse(_fields1[0]);
                            ln.y1 = double.Parse(_fields1[1]);
                            ln.x2 = double.Parse(_fields2[0]);
                            ln.y2 = double.Parse(_fields2[1]);
                            lines.Add(ln);
                            slc = setmaxmin(slc, ln);
                            alllines.Add(ln);
                        }
                    }
                    cline _ln = new cline();//last point closes with
first point
                    string[]   _fields1_   =   fields[fields.Length   -
1].Split(',');
                    string[] _fields2_ = fields[0].Split(',');
                    _ln.x1 = double.Parse(_fields1_[0]);
                    _ln.y1 = double.Parse(_fields1_[1]);
                    _ln.x2 = double.Parse(_fields2_[0]);
                    _ln.y2 = double.Parse(_fields2_[1]);
                    lines.Add(_ln);
                    cntr.lines = lines;
                    cntrs.Add(cntr);
                    slc = setmaxmin(slc, _ln);
                    alllines.Add(_ln);
                    // setminmax3(slc, lines);
```

```
                    //if (v == 2) break;
                }

                slc.contours = cntrs;
                slc.id = svg.g[h].id.Replace("layer", "");

                slice2 wslc = slc;
                double resy = double.Parse(yres);/*y resolution*/
                double resx = double.Parse(xres);/*x resolution*/
                double ytotalrays = (wslc.maxy - wslc.miny) / resy;
                double y = wslc.miny;
                double xtotalrays = (wslc.maxx - wslc.minx) /
resx;/*added*/
                double x = wslc.minx;/*added*/
                wslc.xintersectionpoints = new List<List<ipoint>>();
                wslc.yintersectionpoints = new List<List<ipoint>>();
                for (int m = (int)Math.Floor(wslc.miny); m <
ytotalrays; m++)
                {
                    cline ryl = new cline();
                    ryl.x1 = wslc.minx;
                    ryl.y1 = y;
                    ryl.x2 = wslc.maxx;
                    ryl.y2 = y;
                    List<ipoint> ips = new List<ipoint>();
                    //if (m == 20)
                    {
                        for (int k = 0; k < wslc.contours.Count;
k++)///for y
                        {
                            for (int j = 0; j <
wslc.contours[k].lines.Count; j++)
                            {
                                float x1 =
(float)wslc.contours[k].lines[j].x1;
                                float y1 =
(float)wslc.contours[k].lines[j].y1;
                                float x2 =
(float)wslc.contours[k].lines[j].x2;
                                float y2 =
(float)wslc.contours[k].lines[j].y2;
                                float ix1 = (float)ryl.x1;
                                float iy1 = (float)ryl.y1;
```

```
                                float ix2 = (float)ryl.x2;
                                float iy2 = (float)ryl.y2;
                                PointF p = FindLineIntersection(new
PointF(x1, y1), new PointF(x2, y2), new PointF(ix1, iy1), new
PointF(ix2, iy2));
                                if (p != PointF.Empty)
                                {
                                    ipoint ip = new ipoint();
                                    ip.x = p.X;
                                    ip.y = p.Y;
                                    ips.Add(ip);
                                }



                            }
                        }
                        ips = ips.OrderBy(o => o.x).ToList();
                        wslc.xintersectionpoints.Add(ips);
                    }
                    y += resy;
                }

                for (int m = (int)Math.Floor(wslc.miny); m <
xtotalrays; m++)////for x
                {
                    cline ryl = new cline();
                    ryl.x1 = x;
                    ryl.y1 = wslc.miny;
                    ryl.x2 = x;
                    ryl.y2 = wslc.maxy;
                    List<ipoint> ips = new List<ipoint>();
                    //if (m == 20)
                    {
                        for (int k = 0; k < wslc.contours.Count; k++)
                        {
                            for (int j = 0; j <
wslc.contours[k].lines.Count; j++)
                            {
                                float x1 =
(float)wslc.contours[k].lines[j].x1;
                                float y1 =
(float)wslc.contours[k].lines[j].y1;
```

```
                                float               x2                =
(float)wslc.contours[k].lines[j].x2;
                                float               y2                =
(float)wslc.contours[k].lines[j].y2;
                                float ix1 = (float)ryl.x1;
                                float iy1 = (float)ryl.y1;
                                float ix2 = (float)ryl.x2;
                                float iy2 = (float)ryl.y2;
                                PointF  p  =  FindLineIntersection(new
PointF(x1,  y1),  new  PointF(x2,  y2),  new  PointF(ix1,  iy1),  new
PointF(ix2, iy2));
                                if (p != PointF.Empty)
                                {
                                    ipoint ip = new ipoint();
                                    ip.x = p.X;
                                    ip.y = p.Y;
                                    ips.Add(ip);
                                }


                            }
                        }
                        ips = ips.OrderBy(o => o.y).ToList();
                        wslc.yintersectionpoints.Add(ips);
                    }
                    x += resx;
                }



                ////render this for thesis result presentation.
                ////render the ray cast lines
                wslc.xintersectionlines = new List<List<cline>>();
                wslc.xsolidlines = new List<List<cline>>();
                wslc.xreserrorlines = new List<List<cline>>();
                wslc.ixreserrorlines = new List<List<cline>>();

                wslc.yintersectionlines = new List<List<cline>>();
                wslc.ysolidlines = new List<List<cline>>();
                wslc.yreserrorlines = new List<List<cline>>();
                wslc.iyreserrorlines = new List<List<cline>>();
                if (wslc.xintersectionpoints.Count > 0)
```

```
                {
                    for (int k = 0; k < wslc.xintersectionpoints.Count;
k++)
                    {
                        List<cline> raylines = new List<cline>();
                        List<cline> solidlines = new List<cline>();
                        List<cline>     soliderrorlines     =     new
List<cline>();
                        List<cline>     isoliderrorlines     =     new
List<cline>();
                        if (wslc.xintersectionpoints[k].Count > 1)
                        {
                            for     (int     v     =     0;     v     <
wslc.xintersectionpoints[k].Count - 1; v++)//all intersections
                            {
                                cline cl = new cline();
                                cl.x1                                   =
wslc.xintersectionpoints[k][v].x;
                                cl.y1                                   =
wslc.xintersectionpoints[k][v].y;
                                cl.x2  =  wslc.xintersectionpoints[k][v
+ 1].x;
                                cl.y2  =  wslc.xintersectionpoints[k][v
+ 1].y;
                                raylines.Add(cl);
                            }

                            wslc.xintersectionlines.Add(raylines);
                            // if (raylines.Count > 1)
                            {
                                for (int w = 0; w < raylines.Count; )
                                {

                                    solidlines.Add(raylines[w]);
                                    if    (Math.Abs(raylines[w].x1    -
raylines[w].x2) < resx)
                                    {

soliderrorlines.Add(raylines[w]);
                                        wslc.totalregionwitherrors   +=
1;
                                    }
                                    w += 2;
                                    int v = w - 1;
                                    if (v < raylines.Count)
```

```
                                    {
                                        if (Math.Abs(raylines[v].x1 -
raylines[v].x2) < resx)
                                        {
                                            var       iraylines       =
raylines[v];
                                            iraylines.x2              =
(iraylines.x2 - iraylines.x1) / 2 + iraylines.x1;

isoliderrorlines.Add(iraylines);
                                            wslc.totalregionwitherrors
+= 1;
                                        }
                                    }
                                }
                            }
                            wslc.xsolidlines.Add(solidlines);
                            wslc.xreserrorlines.Add(soliderrorlines);
                            wslc.ixreserrorlines.Add(isoliderrorlines);
                        }
                    }
                }

                if (wslc.yintersectionpoints.Count > 0)
                {
                    for (int k = 0; k < wslc.yintersectionpoints.Count;
k++)
                    {
                        List<cline> raylines = new List<cline>();
                        List<cline> solidlines = new List<cline>();
                        List<cline>     soliderrorlines     =     new
List<cline>();
                        List<cline>     isoliderrorlines     =     new
List<cline>();
                        if (wslc.yintersectionpoints[k].Count > 1)
                        {
                            for     (int     v     =     0;     v     <
wslc.yintersectionpoints[k].Count - 1; v++)//all intersections
                            {
                                cline cl = new cline();
                                cl.x1                                  =
wslc.yintersectionpoints[k][v].x;
                                cl.y1                                  =
wslc.yintersectionpoints[k][v].y;
```

```
                                cl.x2  =  wslc.yintersectionpoints[k][v
+ 1].x;
                                cl.y2  =  wslc.yintersectionpoints[k][v
+ 1].y;
                                raylines.Add(cl);
                            }

                            wslc.yintersectionlines.Add(raylines);
                            // if (raylines.Count > 1)
                            {
                                for (int w = 0; w < raylines.Count; )
                                {
                                    solidlines.Add(raylines[w]);
                                    if    (Math.Abs(raylines[w].y1    -
raylines[w].y2) < resy)
                                    {

soliderrorlines.Add(raylines[w]);
                                        wslc.totalregionwitherrors   +=
1;
                                    }
                                    w += 2;

                                    int v = w - 1;
                                    if (v < raylines.Count)
                                    {
                                        if  (Math.Abs(raylines[v].y1  -
raylines[v].y2) < resy)
                                        {
                                            var       iraylines       =
raylines[v];
                                            iraylines.y2              =
(iraylines.y2 - iraylines.y1) / 2 + iraylines.y1;

isoliderrorlines.Add(iraylines);

isoliderrorlines.Add(iraylines);
                                            wslc.totalregionwitherrors
+= 1;
                                        }
                                    }
                                }
                            }
                            wslc.ysolidlines.Add(solidlines);
                            wslc.yreserrorlines.Add(soliderrorlines);
```

```
                        wslc.iyreserrorlines.Add(isoliderrorlines);
                    }
                }
            }


            wslc.height = wslc.height + 15;
            wslc.width = wslc.width + 25;
            _j.Data = wslc;
        }
        catch (Exception er)
        {
            _j.Data = "Error: " + er.Message;
        }
        finally
        {
            if (reader != null)
                reader.Close();
        }

        return _j;
    }


    private static void setminmax3(slice2 slc, List<cline> lines)
    {
        List<cline> _linex1 = lines.OrderBy(o => o.x1).ToList();
        List<cline> _linex2 = lines.OrderBy(o => o.x2).ToList();
        List<cline> _liney1 = lines.OrderBy(o => o.y1).ToList();
        List<cline> _liney2 = lines.OrderBy(o => o.y2).ToList();
        if (slc.minx > _linex1[0].x1) slc.minx = _linex1[0].x1;
        if (slc.miny > _liney1[0].y1) slc.miny = _liney1[0].y1;
        if (slc.maxx < _linex2[_linex2.Count - 1].x2) slc.maxx =
_linex2[_linex2.Count - 1].x2;
        if (slc.maxy < _liney2[_liney2.Count - 1].y2) slc.maxy =
_liney2[_liney2.Count - 1].y2;

        if (slc.minx > _linex2[0].x2) slc.minx = _linex2[0].x2;
        if (slc.miny > _liney2[0].y2) slc.miny = _liney2[0].y2;
        if (slc.maxx < _linex1[_linex1.Count - 1].x1) slc.maxx =
_linex1[_linex1.Count - 1].x1;
```

```
            if (slc.maxy < _liney1[_liney1.Count - 1].y1) slc.maxy =
_liney1[_liney1.Count - 1].y1;
        }



        public PointF FindLineIntersection(PointF start1, PointF end1,
PointF start2, PointF end2)
        {
            float denom = ((end1.X - start1.X) * (end2.Y - start2.Y))
- ((end1.Y - start1.Y) * (end2.X - start2.X));
            //  AB & CD are parallel
            if (denom == 0)
                return PointF.Empty;
            float numer = ((start1.Y - start2.Y) * (end2.X - start2.X))
- ((start1.X - start2.X) * (end2.Y - start2.Y));
            float r = numer / denom;
            float numer2 = ((start1.Y - start2.Y) * (end1.X -
start1.X)) - ((start1.X - start2.X) * (end1.Y - start1.Y));
            float s = numer2 / denom;
            if ((r < 0 || r > 1) || (s < 0 || s > 1))
                return PointF.Empty;
            // Find intersection point
            PointF result = new PointF();
            result.X = start1.X + (r * (end1.X - start1.X));
            result.Y = start1.Y + (r * (end1.Y - start1.Y));
            return result;
        }
        private slice2 setmaxmin(slice2 slc, cline ln)
        {
            if (slc.minx > ln.x1) slc.minx = ln.x1;
            if (slc.miny > ln.y1) slc.miny = ln.y1;
            if (slc.maxx < ln.x1) slc.maxy = ln.x1;
            if (slc.maxy < ln.y1) slc.maxy = ln.y1;

            if (slc.minx > ln.x2) slc.minx = ln.x2;
            if (slc.miny > ln.y2) slc.miny = ln.y2;
            if (slc.maxx < ln.x2) slc.maxx = ln.x2;
            if (slc.maxy < ln.y2) slc.maxy = ln.y2;
            return slc;
        }
```

Abbreviations

AM	Additive manufacturing
3D	Three dimensional
2D	Two dimensional
dpi	Dots per inch
CAD	Computer Aided Design
FFF	Fused Filament Fabrication
FDM	Fused Deposition Model
EBFF	Electrom Beam Freeform Fabrication
EBM	Electron Beam Melting
MRI	Magnetic Resonnance Imaging
CT	Computed Tomography
RP	Rapid Prototyping
3DP	Three dimensional printing
LM	Layered Manufacturing
SLA	Stereolithography
SLS	Selective Laser Sintering
DMLS	Direct Metal Laser Sintering
ABS	Acrylonitrile butadiene styrene
STL	Standard Tessellation Language
IGES	Initial Graphics Exchange Specification
OBJ	Wavefront's Object File
PLY	Polygon File
STEP	Standard Graphics Exchange Format
SVG	Scalable vector graphics