Development of a 3D Bioprinting Software Toolchain

by

Tamer Abdullah Gharieb Mohamed

B.A.Sc., The University of British Columbia, 2011

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

August 2014

© Tamer Abdullah Gharieb Mohamed, 2014

Abstract

In recent years, three-dimensional (3D) printers have revolutionized the process of prototyping and manufacturing inanimate objects. Extending this technology to tissue engineering as a means of creating customized *in vitro* tissue constructs that mimic *in vivo* conditions is a relatively new idea that has the potential to transform the way biological research is conducted. Biological tissues are inherently complex 3D heterogeneous structures. Many of these tissues are made up of building blocks that vary in composition and morphology. These building blocks are organized into different levels and locations which allow them to interact with one another in unique ways such that the overall tissue structure exhibits a specific biological function. Designing and then printing 3D biological structures composed of multiple cell-encapsulated building blocks, each programmed by composition and architecture and printed using different properties, is a challenge in tissue engineering.

This thesis presents the development of a 3D bioprinting software toolchain for the design and printing of software-programmable tissues. The 3D bioprinting software toolchain is built around a novel bottom-up tissue engineering design method. The Tissue Building Block Design (TBBD) method seeks to enable the assembling of complex biological structures from a set of simpler building blocks, each coded with unique material compositions, printing properties, and architectures. Algorithms were developed to generate the layer-by-layer heterogeneous process plans required to 3D

print tissue models designed using the TBBD method. We evaluate the performance of our implementation of the TBBD method by analyzing execution times and performing a comparison against a more standard design approach. We then analyze and discuss the effect of design choices and printing parameters on the overall printing process and the challenges associated with our microfluidics-based method of bioprinting. We also demonstrate the functionality and asses the capabilities of the 3D bioprinting software toolchain by printing several different heterogeneous hydrogel structures using our 3D bioprinter.

Preface

The printhead described in Chapter 2 was designed and fabricated by Simon Beyer. The 3D printing results presented in Chapter 3 were obtained from experiments that were conducted in collaboration between Simon Beyer and myself. Biological experiments presented in Chapter 3 were conducted by me, Simon Beyer, Sheng Pan, and Samuel Wadsworth. The publications related to this thesis are listed below:

1. Refereed publication: Simon Beyer, **Tamer Mohamed**, and Konrad Walus. "A microfluidics based 3D bioprinter with on on-the-fly multimaterial switching capability." *The 17th International Conference on Miniaturized Systems for Chemistry and Life Sciences,* 2013, Freiburg, Germany.
2. Non-refereed publication: **Tamer Mohamed**, Simon Beyer, and Konrad Walus. "A microfluidics based 3D bioprinter." *CMC TEXPO symposium and competition,* 2013, Gatineau, Quebec (short paper and presentation, reviewed by abstract).

For the first publication, I developed the software required to design and print the multi-material hydrogel structures, collaborated with Simon Beyer to obtain the 3D printing results, and wrote the portion of the manuscript pertaining to the software. For the second publication, I wrote most of the submitted abstract and also presented at the symposium. ***This presentation was awarded first place in the national design competition (CMC MEMSCAP 2013) for demonstrating the greatest degree of novelty and industry-relevance.***

In addition to publications, some of the work presented from this thesis has been submitted in a US provisional patent (US 61/834,420) as well as a PCT patent application (PCT/CA2014/050556). For the provisional patent application, I wrote parts pertaining to the software components of our 3D bioprinting system. Results from my work appear in both the provisional patent and PCT application.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

3D	Three Dimensional
2D	Two dimensional
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
STL	StereoLithography
FFF	Fused Filament Fabrication
CNC	Computer Numerical Control
NC	Numerical Control
DMC	Digital Motion Controller
RTOS	Real Time Operating System
TBBD	Tissue Building Block Design
PTDL	Printed Tissue Description Language
GUI	Graphical User Interface
XML	Extensible Markup Language

Acknowledgements

I would like to acknowledge the unconditional love and support that I've received from my parents. They have both sacrificed everything for me and my siblings. Nothing I do or say can ever repay them.

I am truly thankful for the support, encouragement, and friendship that I've received from my advisor, Dr. Konrad Walus. His wealth of scholarly knowledge and creative problem solving skills has helped me develop into the researcher that I am. It has truly been an honor and privilege to work under his supervision. I also thank him for his generous financial support in the form of research assistantship. I look forward to continuing this relationship as I strive towards my Ph.D. studies.

I would also like to thank my lab mates Simon Beyer, Anas Bsoul, Faizal Karim, Sheng Pan, and Christoph Sielmann for all the thought provoking conversations and for making graduate school fun.

Dedication

This thesis is dedicated to my mother and father who constantly remind me about the importance of knowledge.

Chapter 1: Introduction

3D printing, also known as additive manufacturing or rapid prototyping, refers to the printing of physical 3D structures from computer-aided design (CAD) models in a layer-by-layer approach [1], [2], [3]. Although the first successful demonstration of additive manufacturing occurred over three decades ago, it is only in recent years that 3D printing started revolutionizing the process of manufacturing mechanical parts [4], [5]. 3D printing has now grown into a more than $2 billion industry and is predicted to top $10.8 billion by 2021 [5]. Improvements in computer technology, reduction in hardware costs, expiration of key patents, open-sourcing and collaborative development of relevant software tools and algorithms, and the emergence of new printing materials significantly contributed to the tremendous rate at which 3D printing technology has progressed [6], [7], [8], [9], [10]. 3D printing is now being used to fabricate fully functional parts suitable for end use and is no longer only reserved for prototyping purposes [11]. 3D printing technology has the potential to drastically reduce product development time and cost-effectively create customized and geometrically complex structures on-demand across a wide variety of different applications [1]. This has captivated the imagination of engineers, scientists, and economists to the point where many have asserted 3D printing technology as the “third industrial revolution” [12].

The creation of biological structures from digital models is a quickly emerging application of 3D printing that could potentially have transformative implications on the

field of tissue engineering. This particular application is now commonly being referred to as 3D bioprinting [13]. Besides the ultimate goal of printing organs for repair or replacement, the use of 3D bioprinting to create functional tissue constructs is a shorter-term application that can provide biologists with a novel tool for examining the pathology of specific diseases and facilitating the discovery of new therapeutics [13].

Today it takes an average of $1-4 billion and 12-14 years to develop a drug [14], [15]. Drugs can fail for a variety of reasons, but in many cases it is due to an incorrect efficacy or toxicity assessment made during pre-clinical analysis. Current pre-clinical testing platforms can take a number of different forms; however, the most ubiquitous are 2D cell cultures and animal models, neither of which fully models the human physiology [16], [17], [18]. 2D cultures can differ significantly from the complex 3D intercellular interactions occurring in organs [19]. Animal models provide a powerful system level perspective but are often poor predictors of the human drug response [20]. Recent studies have demonstrated that 3D cell co-cultures better represent human *in vivo* conditions compared to the standard 2D cultures currently being used in the drug discovery process [16], [17], [18]. Moreover, 3D cell co-cultures have the potential to replace animal models. Using 3D co-cultures in the drug screening process could contribute to significant improvements in the predictive accuracy of the pre-clinical drug discovery process by providing the pharmaceutical industry with 3D tissue models that better mimic *in vivo* conditions. These enhancements could drive a fundamental shift in the pharmaceutical industry, enabling the development of completely new therapeutics,

and enabling pharma to test drugs they may have shelved in the past due to a lack of appropriate models.

This thesis is comprised of four chapters:

- Chapter 1: An introduction to 3D printing, its application to creating biological structures, the general 3D printing software workflow, and a review of the most related literature.
- Chapter 2: An overview of our developed 3D bioprinting system. Then detailed explanations of the design and implementation of the methods and core algorithms used in the 3D bioprinting software toolchain are presented.
- Chapter 3: An analysis and discussion of observed results from test simulations and 3D printing experiments that were conducted to evaluate the performance, test and verify the functionality of our developed algorithms, and assess the effect of design choices and printing parameters on the overall printing process. Results from 3D printing hydrogel structures are also presented and discussed in this chapter.
- Chapter 4: Concluding remarks and suggestions on future directions of this project.

1.1 Motivation

The ability to rapidly design and then print 3D heterogeneous human tissues on-demand could enable advancements in the fields of tissue engineering, drug discovery,

and regenerative medicine [13]. Biological tissues are inherently complex heterogeneous structures made up of different regions. Each region may be composed of several key components such as cells, extracellular matrices, and intricate vascular networks that are arranged in specific 3D geometries [13]. For example, the anatomy of an artery (shown in Figure 1) consists of a central lumen and a set of concentric walls with different architectures, cell compositions, and morphologies. In order to engineer a physiologically relevant tissue that mimics native behavior, different regions of a structure may need to be printed using different properties. Control over composition, structure, and parameters such as patterning and porosity is essential as these properties have been shown to affect biological response [13], [21], [22].

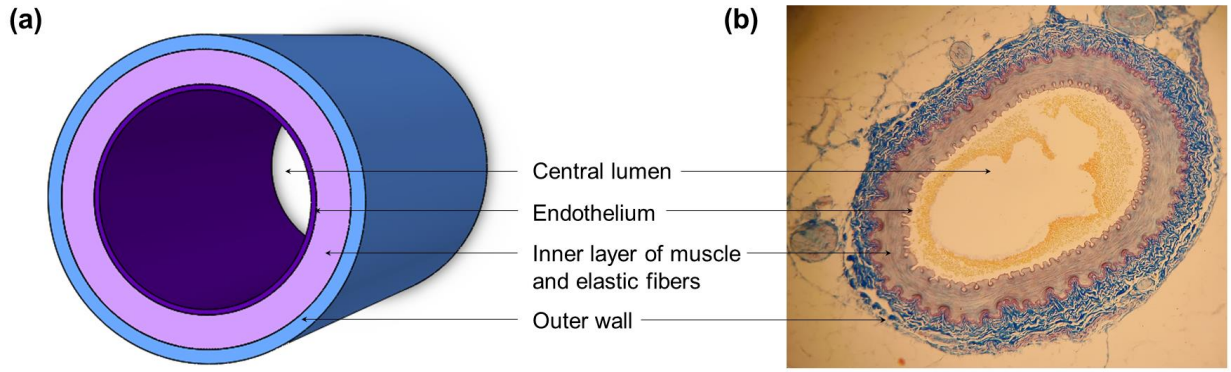


Figure 1: *(a) 3D CAD model of an artery (b) Cross-Section of artery (source: Wikimedia Commons*)

Designing and then printing cell-laden structures with complex geometries, multiple materials, heterogeneous toolpath planning and printing properties in an automated way presents a challenge in tissue engineering. Providing new design

methods and tools that enable the creation of 3D tissue models as well as a platform for sharing the software-described tissue models is expected to aid biologists and tissue engineers in advancing the field of *in vitro* tissue model development.

1.2 Objectives

The primary goals of this work are to develop an automated process and user-friendly software toolchain for designing and 3D printing heterogeneous tissue structures as well as a platform for describing and sharing tissue models. Achieving the objectives of this work will first require an in-depth understanding of standard 3D printing principles and a study of the most common file formats and open source software tools currently being used by the 3D printing community. Then a method will be developed for designing 3D biological structures, taking into account the inherent heterogeneity and complexity of native tissues. After that, algorithms will be designed and implemented to code tissues in 3D with programmable geometrical and material properties. Then the performance of the developed algorithms will be evaluated by analyzing execution times. Software will then be developed to facilitate control and coordination of printhead function and stage motion. A language for describing and sharing tissue models will be developed. The effect of design choices and printing properties on the overall printing process will be analyzed and discussed. To verify the functionality and demonstrate the capabilities of the developed software toolchain, several test structures will be designed and then 3D printed.

1.3 3D Printing

The basic idea behind 3D printing is that a CAD object is translated into a physical structure by consecutively stacking a set of 2D layers until the entire structure is realized. Each deposited 2D layer represents a cross-sectional slice of the 3D structure and has an associated finite layer thickness. The 3D printing process thus yields structures that are approximations of CAD models where a layer is the basic building unit [1]. Reducing the layer thickness improves the quality of the resulting 3D structure at the cost of increasing the time required to build the entire structure [1]. Within each layer, various printing control parameters are manipulated in order to vary the stability, porosity, and composition of the printed structure to suit the specific application requirements.

Of the many different 3D printing deposition methods, currently one of the most common processes is an extrusion-based technology known as Fused Filament Fabrication (FFF) [1]. FFF was developed and commercialized by Stratasys, Ltd. more than two decades ago [23]. The way FFF works is that a solid thermoplastic filament is heated into a liquid polymer that is extruded through a nozzle and patterned according to a computer-controlled motion sequence [23].

In contrast to 3D printers, which use an additive manufacturing process, subtractive manufacturing machines such as Computer Numerical Controlled (CNC) machines start with a solid block of material and use cutting tools to sculpt the desired object by removing the excess [24]. Because of this fundamental difference, 3D printing

poses several benefits over subtractive manufacturing processes. Some examples of these benefits include a reduction in material consumption, increased geometrical complexity doesn't necessarily complicate the fabrication process, and greater simplification in programming and software preparation of build instructions [1], [25]. A diagram depicting the difference in methodology between additive manufacturing and subtractive manufacturing is shown in Figure 2.

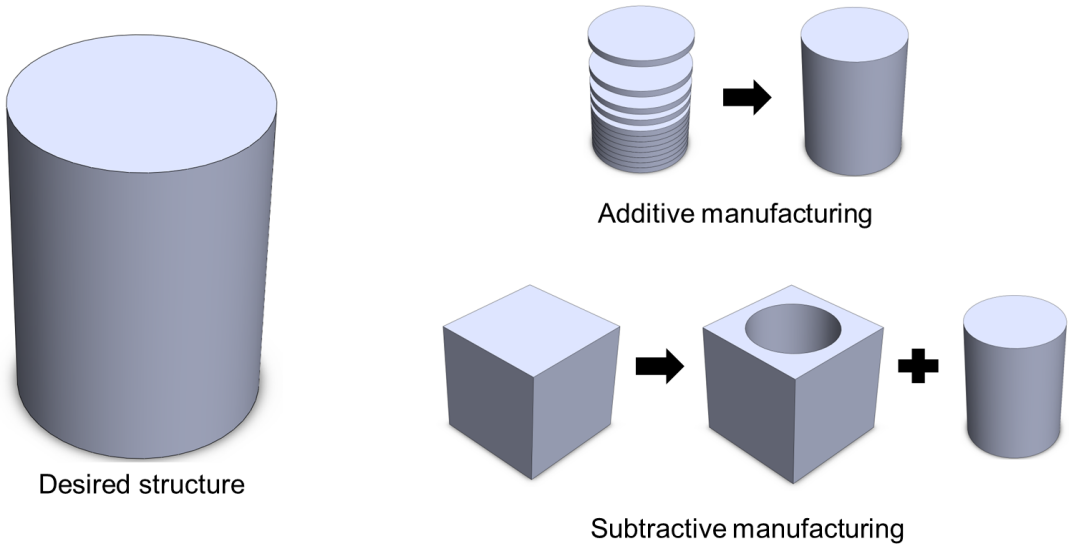


Figure 2: *Additive Manufacturing versus subtractive manufacturing*

With the rapid decrease in the cost of 3D printers in recent years, a thriving online community of 3D printing enthusiasts has emerged. Through this open community, a general 3D printing toolchain has been adopted and users actively participate in the collaborative development of different computer-aided design (CAD) and computer-aided manufacturing (CAM) software tools, such as OpenSCAD,

ReplicatorG, Slic3r, and Printrun [3]. Initiatives such as the RepRap and Fab@Home projects provide complete hardware specifications for users to construct fully working personal 3D printers [9], [26]. Websites such as Thingiverse serve as platforms for hosting and sharing 3D printing design files [27].

The generally accepted 3D printing software workflow for transforming a 3D model in the virtual world to its real world equivalent consists of three steps: modeling, slicing, and then printer control to print the actual structure [1], [3]. While the specific implementation may vary slightly and extra elements such as initialization procedures and post-processing functions may be incorporated for different printing applications, the software workflow and the associated file formats shown in Figure 3 are common across almost all 3D printers. Modeling involves fully specifying the surface geometry of the desired 3D structure using CAD software such as SolidWorks. The surface geometry is most commonly described and exported as a StereoLithography (STL) file. Slicing involves the use of computer-aided manufacturing (CAM) tools to translate a 3D CAD model into machine instructions understood by the particular 3D printer used in the printing process. The generated slicing data is commonly described using G-Code [3]. Printer control is the process whereby machine-specific commands are communicated to the 3D printer in order to create the entire structure.

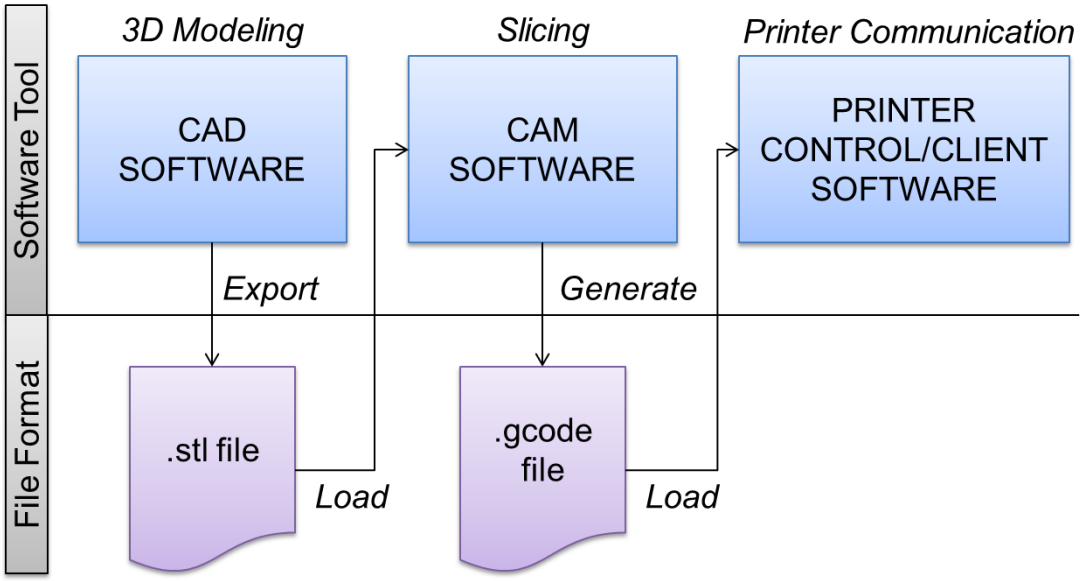


Figure 3: *Software tools and file formats used in the general 3D printing software toolchain*

The following sections will describe the details behind the different software elements and file formats used in the general 3D printing process. The first section explains the process of designing a computer model of a desired physical structure using CAD software and provides a background on the industry-standard tessellation file format. The second section describes the process of generating toolpaths required to realize a desired physical structure using CAM software and provides an overview of one the most commonly used numerical control languages.

1.3.1 3D Model Design and Tessellation

Product design is the first step in any systematic manufacturing process and involves conceptualizing an idea and transforming it into a fully defined visualization [28]. This also applies to the 3D printing process, which usually starts with designing a 3D computer model of a physical structure. Models can be designed by a user through the use of 3D CAD modeling software, obtained or modified from online sharing repositories, or reverse-engineered using application specific technologies such as 3D scanning or medical imaging. The output of this step is a 3D model that is most commonly described and represented using the STL file format [1], [3].

Created by 3D Systems Inc. in 1987, the STL file format has since become the industry standard and is currently supported by almost all 3D CAD software programs [29], [30], [31]. STL is a tessellated representation of a physical structure whereby the surface geometry of a 3D model is approximated by a list of triangular facets without attributes such as material information and CAM specifications [29], [30], [31]. Furthermore, no measurement units are assigned in the STL file, so software programs using STL files usually allow users to specify the desired units [30]. Each of the triangular facets is described by a unit normal and three vertices as shown in Figure 4 [29], [30], [31]. The unit normal and each of the three vertices are defined by X, Y, and Z coordinates.

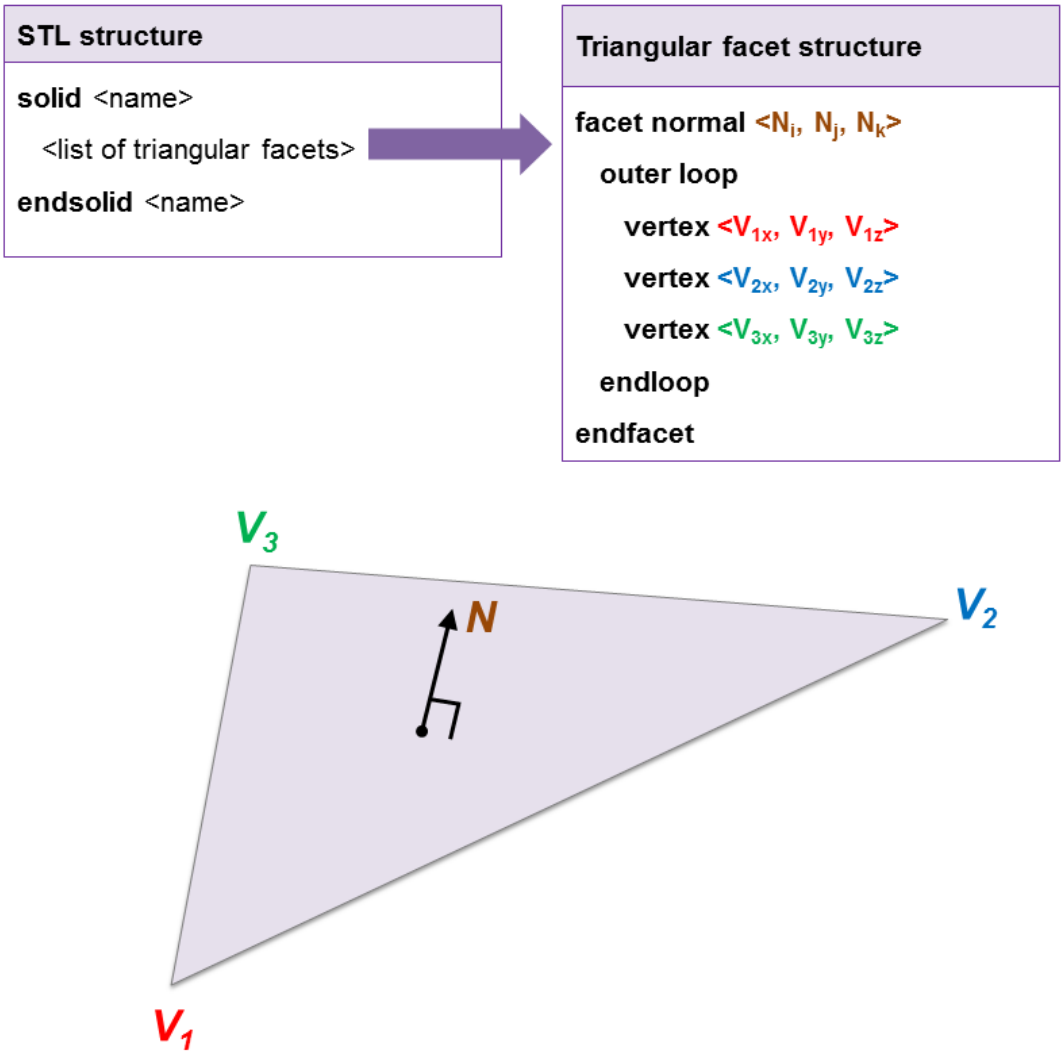
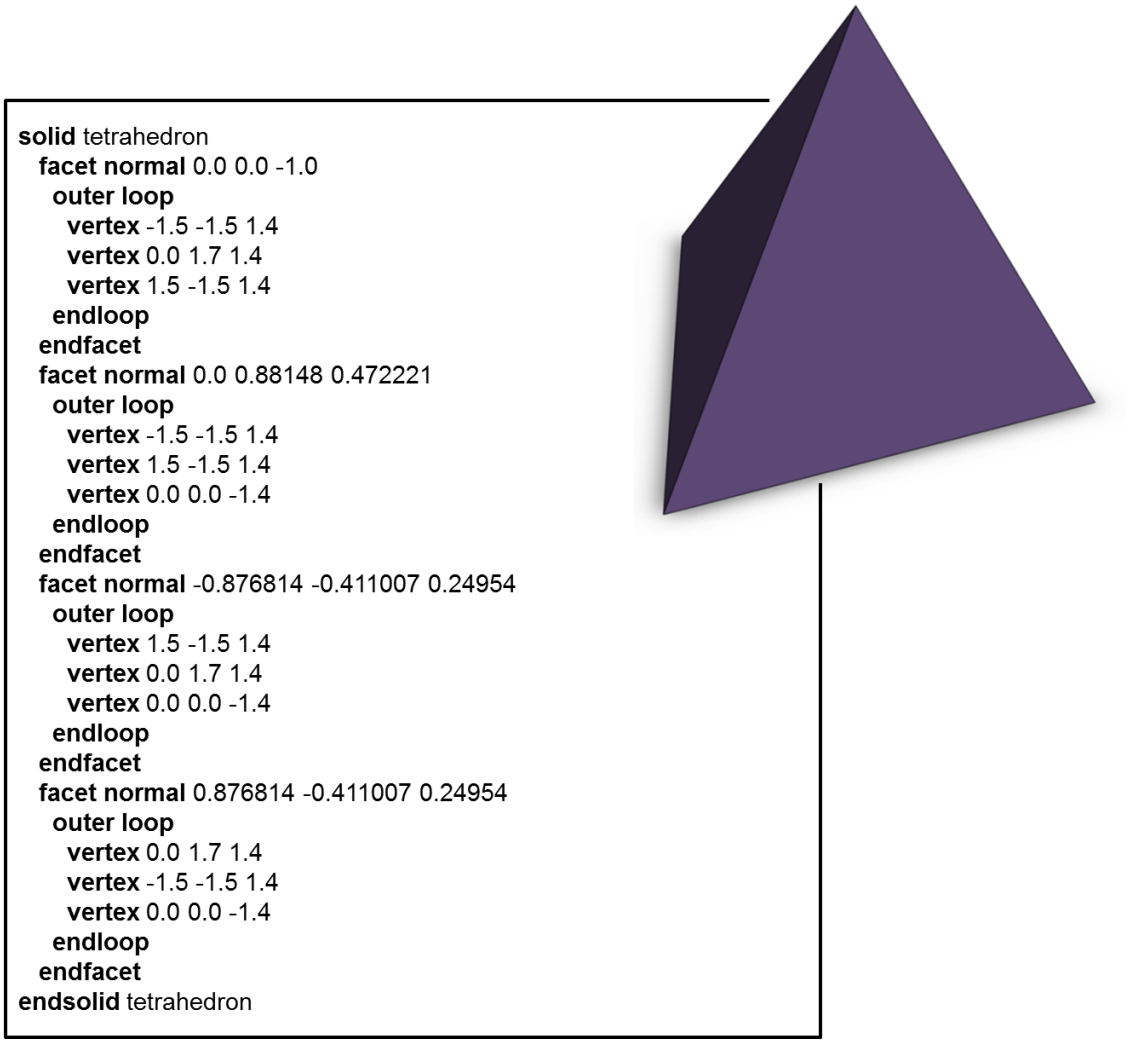


Figure 4: STL file structure

STL files come in two different storage formats: ASCII and binary. Both formats use a ".STL" extension [29]. The ASCII format is human-readable and is mostly reserved for debugging and instructional purposes while the binary format is more commonly used due to its compact file size. An example of an ASCII STL file exported

using SolidWorks that describes a tetrahedron is shown in Figure 5. Although both the ASCII and binary versions approximate the tetrahedron using 4 triangular facets, the file sizes are 722 bytes and 284 bytes, respectively.



```
solid tetrahedron
  facet normal 0.0 0.0 -1.0
    outer loop
      vertex -1.5 -1.5 1.4
      vertex 0.0 1.7 1.4
      vertex 1.5 -1.5 1.4
    endloop
  endfacet
  facet normal 0.0 0.88148 0.472221
    outer loop
      vertex -1.5 -1.5 1.4
      vertex 1.5 -1.5 1.4
      vertex 0.0 0.0 -1.4
    endloop
  endfacet
  facet normal -0.876814 -0.411007 0.24954
    outer loop
      vertex 1.5 -1.5 1.4
      vertex 0.0 1.7 1.4
      vertex 0.0 0.0 -1.4
    endloop
  endfacet
  facet normal 0.876814 -0.411007 0.24954
    outer loop
      vertex 0.0 1.7 1.4
      vertex -1.5 -1.5 1.4
      vertex 0.0 0.0 -1.4
    endloop
  endfacet
endsolid tetrahedron
```

Figure 5: *ASCII STL File describing a tetrahedron*

1.3.2 3D Model Slicing and Toolpath Generation

Following the completion of the necessary modeling and tessellating tasks using 3D CAD software and exporting the corresponding STL file, the next step in the 3D printing workflow involves CAM process planning. This includes generating the necessary printer instructions and sending these instructions to the 3D printer.

Since 3D printers build up physical objects through the successive deposition of material in a layer-wise approach, tessellated objects described in STL files must first be sectioned into a set of geometrically and dimensionally correct 2D horizontal layers (slices) with associated user-defined layer thicknesses. This process is commonly referred to as slicing within the 3D printing community.

The slicing process is achieved by first intersecting the 3D model with a set of imaginary slicing planes whose common normal is in the vertical build direction (as shown in Figure 6). From the intersection points that are formed between a given imaginary slicing plane and the 3D surface representation, a contour (perimeter) is drawn out for each slice. After the perimeters are defined for each slice, toolpaths that the printhead needs to follow in order to fill the space enclosed by each slice's perimeter with raw material according to a specific pattern are generated. The output of the slicing process is printing instructions.

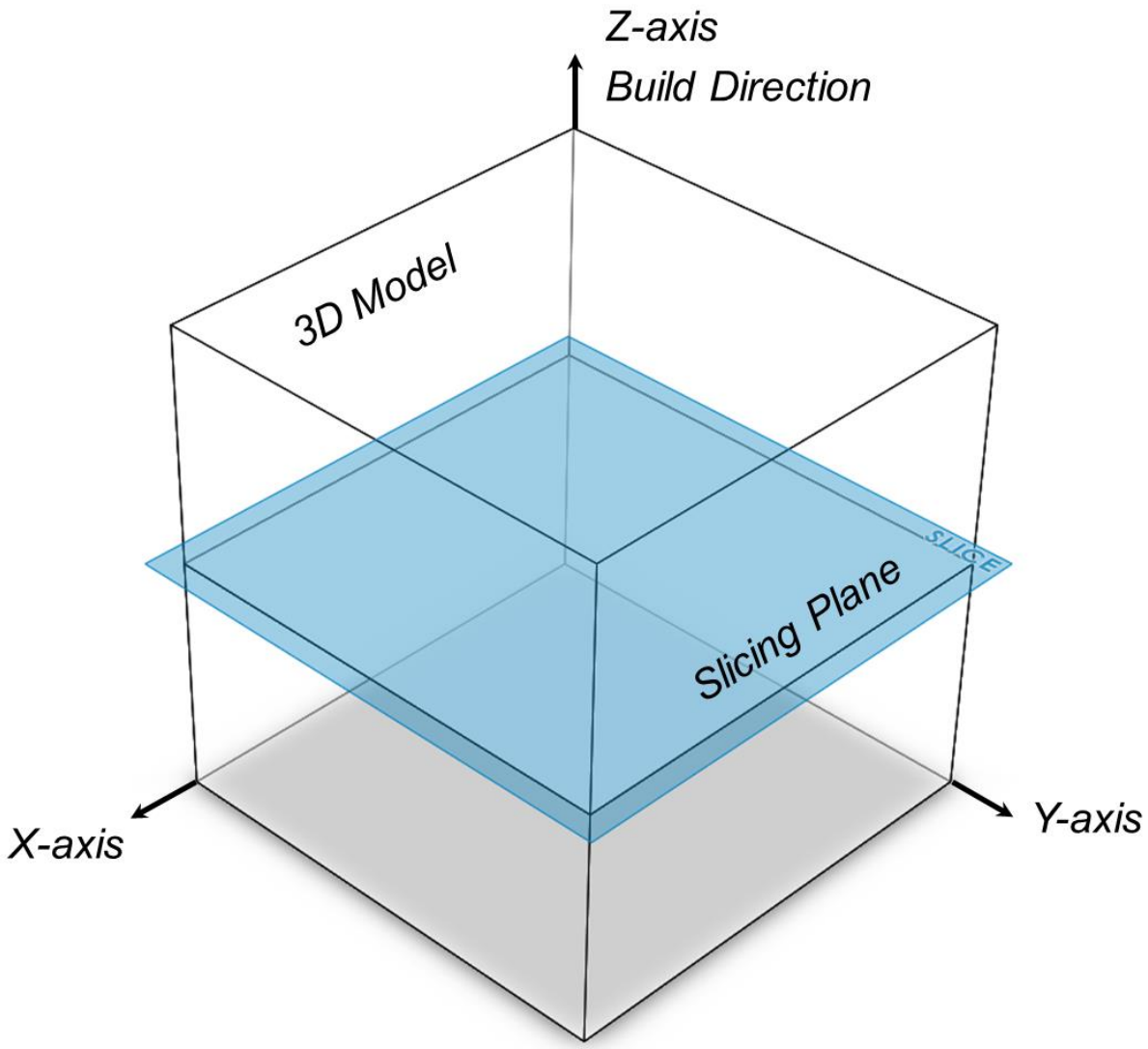


Figure 6: *Intersection of slicing plane and 3D model*

The entire slicing process, also sometimes referred to as toolpath generation, can be accomplished through the use of slicer software. With the rise of 3D printing, several different slicer applications have been collaboratively developed, optimized, and are currently available online as open-source applications. Some examples of the more

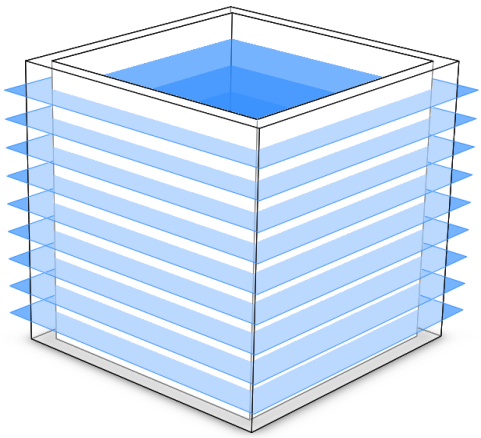
common slicer programs include Slic3r, Skeinforge, Repsnapper, and Cura. Slicer applications typically accept an STL file as an input and allow the user to define several different CAM specifications such as printhead speed, layer thickness, and fill density and patterns in order to generate the given set of printing instructions. Some slicer applications employ additional algorithms to add extra fill, perimeters, and layers when needed in order to ensure structural stability and improve the quality of the printed object. Given a specific STL input file, the output of the slicer software is a file containing a set of printing instructions described using numerical control (NC) code. An NC program is essentially a structured sequence of code blocks specifying the exact motion profiles and supplementary operations to be executed by the 3D printer in order to create a physical structure [32].

G-Code is the most widely used NC Programming language, used extensively over the past few decades for CNC part programming, and more recently for desktop 3D printing toolpath generation [3]. For this reason, many open source slicer applications are sometimes also referred to as G-Code generators. Although a specific version of G-Code was standardized in the 1980s with ISO 6983 [33], there currently exist various different implementations independently developed by motion controller manufactures [34], [35]. Moreover, some controllers don't understand G-Code. This variability in motion controllers makes it difficult for a single G-Code program to drive different motion controllers. Furthermore, many additive manufacturing machines use several unique functions that make it difficult to standardize a specific NC language. Due to the lack of a unified control language understood by all controllers, in many

cases where G-Code generators are used for slicing, G-Code post-processing or interpretation is a necessary step in order to ensure the code is correctly understood by the machine [34], [35].

The G-Code programming language is built upon sequential lines of code called blocks [36]. Each G-Code block is comprised of a series of words [36]. Each word is comprised of a letter followed by a numerical value [36]. Words are used to denote specific machine instructions or arguments. The complete list of allowable letters and numerical precision is determined by the controller manufacturer. A G-Code program is a file that consists of a list of blocks ordered in a structured way to describe the exact motion and auxiliary functions that a machine must perform [36]. An example of a G-Code program describing the motion toolpaths required to 3D print a 10 mm cube with no top or bottom is shown in Table 1. The G-Code program consists of 50 slices (layer thickness of 0.2 mm) in the XY-plane and the corresponding new slice movements in the Z-axis. The toolpath of the first slice is shown in Figure 7.

Table 1: *G-Code program describing the toolpath of a cube with no top or bottom*

3D Model Sliced		
		
G-Code Program		
#	**G-Code Block**	**Description**
1	G28	Home all axes
2	G90	Use absolute coordinates
3	G21	Set unit to millimeters
4	G1 Z0.200 F1800.000	Move to next layer (0)
5	G1 X5.050 Y14.950	Move to first perimeter point
6	G1 X5.050 Y5.050	Perimeter point
7	G1 X14.950 Y5.050	Perimeter point
8	G1 X14.950 Y14.950	Perimeter point
9	G1 X5.050 Y14.950	Perimeter point
10	G1 Z0.400	Move to next layer (1)
11	G1 X5.050 Y5.050	Perimeter point
12	G1 X14.950 Y5.050	Perimeter point
13	G1 X14.950 Y14.950	Perimeter point
14	G1 X5.050 Y14.950	Perimeter point
.	.	.
.	.	.
.	.	.
250	G1 Z10.000	Move to next layer (49)
251	G1 X5.050 Y5.050	Perimeter point
252	G1 X14.950 Y5.050	Perimeter point
253	G1 X14.950 Y14.950	Perimeter point
254	G1 X5.050 Y14.950	Perimeter point

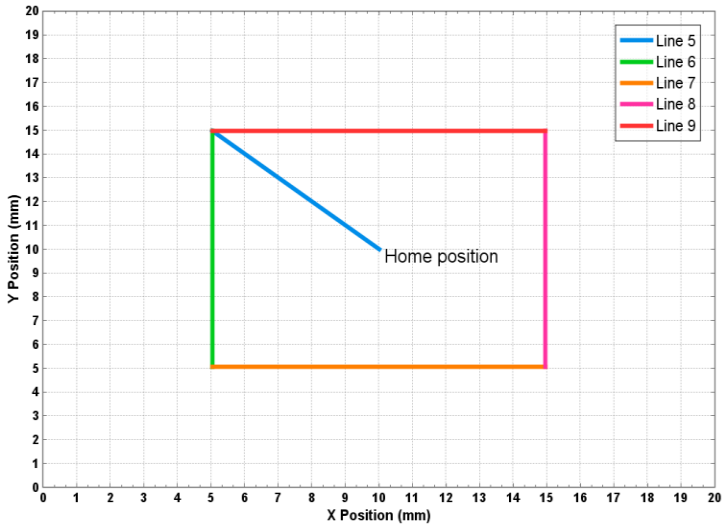


Figure 7: *Toolpath of first layer of sliced cube with no top or bottom*

In the example G-Code program shown in Table 1, the first few commands (lines 1-3) are preparatory G-Code instructions generated for a particular 3D printer hardware configuration. These preparatory instructions are included to perform functions such as initializing all axes and configuring the appropriate units. The remaining lines of code (lines 4-254) are all G1 commands. G1 commands are the most commonly used commands generated by open source slicer programs. The G1 command denotes a straight line movement from the current point to the specified point at a given speed. For example, on line 4 in Table 1, the G1 command instructs the printer to move in a straight line from the current position to the specified Z coordinate (0.200 mm) at the programmed speed (1800 mm/minute) in order to start printing a new layer of material. On line 5, the G1 command instructs the printer to move in a straight line from the current position to the X and Y coordinates (5.050 mm, 14.950 mm) which corresponds to the first perimeter point.

After successfully designing the model of the desired physical structure using 3D CAD software, exporting the resulting STL tessellation file, slicing the model and generating the corresponding G-Code toolpath file, the last step in the 3D printing software workflow involves sending the G-Code file to the 3D printer via some communication link such as USB or Ethernet. This communication task is handled by the 3D printer's client (printer control) software. The printer control software sends the sequence of instructions to the firmware installed on the 3D printer's microcontroller to execute a particular printing process. The 3D printer control software usually provides a graphical user interface (GUI) with printing-related functions such as initialization, direct motion control, and printing start/stop. The printer firmware interprets the received instructions and prints the desired structure.

1.4 Related Work

There are currently two main methods commonly used in tissue engineering [37]. The first method is a top "top-down" approach which involves seeding cells onto a biocompatible or biodegradable structure called a scaffold [37]. The cell-seeded scaffold structure is then incubated and cultured until the cells grow and start to mimic real human tissue [37]. The second method is a "bottom-up" approach which involves using simple cell-laden building blocks to build up more complex tissue pre-cursors [37]. These tissue pre-cursors must then be cultured to allow for the cells to differentiate and eventually exhibit a physiologically-relevant function. Bottom-up tissue engineering methods could produce tissues that better represent *in vivo* conditions, especially for

biological structures that have repetitive units such as the liver [38]. Fiber-based printing is one example of a bottom-up tissue engineering approach whereby cell-laden fibers are patterned into structures [38]. This approach to tissue engineering is analogous to the popular FFF methods used in many commercial 3D printing systems such as the MakerBot Replicator Desktop 3D Printer, RepRap, and others.

Most bioprinting systems rely on two types of printing technologies: inkjet printing [39] and extrusion printing [40]. Inkjet-based bioprinting systems allow for high-resolution and micro-scale printing [41], [42]. Some problems related to inkjet-based bioprinting include high mechanical shear of ejected cells, clogging at the orifice, limited to printing materials with low viscosities, sedimentation of cells in inkjet cartridge, and lengthy build times required to print macro-scale structures [41], [42]. While extrusion-based bioprinting systems enable the printing of large structures, they are typically limited to printing macro-scale structures and materials with sufficiently high viscosities [41], [42]. Furthermore, most extrusion-based systems rely on a separate nozzle for each material.

The bioprinting system developed in this work uses a microfluidic printhead to deposit a hydrogel fiber that is chemically cross-linked on-chip via coaxial flow focusing. Using this approach, material is loaded into the printer as a liquid, but dispensed as a solid fiber, allowing for various on-chip programmatic microfluidic operations prior to fiber formation. Generating cell-laden hydrogel fibers using this approach, commonly referred to as microfluidic spinning, has been reported by several others [43], [44], [45]. Extending this concept to 3D printing for the fabrication of fiber-based biological

structures has also been demonstrated before [46], but with very limited automation and without the ability to design complex heterogeneous structures based on user-defined 3D CAD models. The work described in [46] relies on manually programming a set of motion instructions, without the use of user-defined 3D CAD models, to print simple patterns. Moreover, while the work in [46] demonstrates the use of multiple materials, material changeovers during the printing process are handled manually and are not automatically triggered based on pre-defined CAD specifications and material information.

Traditionally, 3D printing systems were used to create objects that are homogenous in terms of material composition and CAM specifications [1], [5], [47]. This explains why STL – which is the most common file format compatible with almost every 3D printing system – only describes the surface geometry of a structure, and does not include any material information or CAM specifications [29], [30], [31]. STLs have been used in many studies focused on printing homogenous biological structures [48], [49]. Open source slicers such as RepRap and Slic3r have been used to generate the toolpaths from an input STL file [50], [51] . In recent years, 3D printers capable of synthesizing multiple material objects have emerged due efforts from industry, academia, and a thriving community of 3D printing enthusiasts [52].

For applications that require the use of multiple materials such as heterogeneous bioprinting, others have developed software that takes multiple STLs as inputs, then assigns material information to each STL, and then compiles the set of STLs into a multi-material model [1], [53], [54]. The multi-material model defined in CAD is then fed

into CAM software in order to generate the set of printing instructions. Using this approach, the same set of CAM specifications is enforced upon the entire structure. This greatly limits the control over print settings and overall functionality of the object. Others have proposed revised STL file formats that include additional information besides just geometry. One example of this is the Additive Manufacturing File Format (AMF) [55]. In addition to specifying an object's geometry, AMF supports composition and color descriptions [55]. However, most 3D CAD modeling programs, such as SolidWorks, are not compatible with AMF. Additionally, using AMF poses the same limitations as the approach of assigning material information to a set of STLs; namely, the same properties are imposed upon the entire model during the CAM step. Other bioprinting systems rely on manually programming instructions line-by-line as well as heavily relying on human intervention to facilitate material changeovers during a print job [38]. There remains a need to develop standard 3D printing design approaches that allow users to easily assign materials and properties to different regions of a 3D model, generate the printing process plans, and then print these heterogeneous structures in an automated way [47].

Though high-end commercial 3D printers, such as those developed by Stratasys and 3D Systems, are starting to support multi-material printing, these systems often rely on proprietary file formats that are tailored to specific printer hardware configurations [47]. Most of these printing systems use inkjet technology and in-house rasterization algorithms to simultaneously deposit multi-material liquid photopolymers that are cured using ultraviolet light. Although these printers are capable of creating objects with high

resolution, building large structures using this approach can be extremely time consuming.

Filament-based multi-material 3D printers, such as those developed my MakerBot and the multi-nozzle bioprinter described in [54] use a dedicated extruder for each material. This type of system configuration usually leads to disruptions, discontinuities, and time delays during the printing process as a result of printhead changeovers [47]. Reducing disruptions caused by material changeovers is still an ongoing challenge [47].

While process planning for single-material 3D printing has been extensively researched over the past two decades, research around processing planning for multi-material systems is still in its infancy [56]. To avoid redundant tool changeovers in multi-nozzle printers, algorithms that organize toolpaths of the same material type at the same level into groups have been investigated [57]. Our software toolchain supports the printing of multi-material structures using a single nozzle printhead. We developed algorithms to support the programmatic switching of materials on-the-the-fly without delays associated with printhead changeovers, allowing for the continuous and synchronized interweaving of programmable multiple material fibers both within and across layers.

Commercial bioprinting software developed by Organovo is described in [58]. The software is capable of fabricating structures composed of various hydrogels and cell types. The software operates in two modes: “Click ‘N Print” and “Script ‘N Print” [41], [58]. In the “Click ‘N Print” mode, a blank cross-section of stacked cylinders is

displayed on a GUI and users can design simple structures by assigning a material type to each of the cylinders [41], [58]. Basing the design process around cylindrical building blocks rather than user-specified 3D CAD models greatly limits patterning capabilities and control over the structure's geometry and composition. The "Script 'N Print" mode is used to design more complex structures, but requires users to manually program movements of the positioning stage and multiple printheads [41], [58]. This process can be extremely time consuming, error-prone, and requires extensive knowledge and experience about motion control programming.

To overcome the lack of a universal G-Code specification, we developed a G-Code interpreter using Unix-based compiler utilities. Interpreting G-Code using similar compiler utilities has been demonstrated for CNC applications [59], but not for 3D printing applications. The G-Code interpreter is used to translate the G-Code generated by the slicer program to machine code understood by the motion controller used in our bioprinting system (Galil DMC-2140). The use of compiler utilities allows for interoperability across different bioprinting systems driven by different motion controllers. While Galil Motion Control, Inc. (Galil) previously developed proprietary software that translates G-Code to Galil's specific machine code (DMC), that application is no longer publicly available and only supports a form of G-Code commonly used in CNC applications and is not directly compatible with 3D printing slicer software. Galil also developed a proprietary CAD-to-DMC program that translates DXF files into DMC commands; however, this software is limited to 2D CAD drawings [60].

In this work, we focus on developing a novel 3D bioprinting software toolchain that is compatible with industry standard file formats and CAD/CAM tools for the fabrication of 3D printed biological structures with heterogeneous properties based on user-defined 3D models. We present a design method that involves assembling complex structures from simpler building blocks; each building block could be programmed and implemented using different properties such as fiber diameter, deposition/infill patterning, porosity, layer thickness, and composition. Using our approach, a set of STL building blocks are sliced independently, according to a unique set of CAM specifications, to generate a set of G-Code files. Then each G-Code file is assigned a material type, 3D coordinates, and other unique properties compatible with multifunctional printhead processes. The set of G-Code files is then merged into a single file in order to generate the heterogeneous layer-by-layer process planning instructions. In this way, the process of assigning material information, print settings, and other unique properties is performed during the CAM step. This enables us to decouple our unique ability to design and print building blocks under different conditions from standard approaches used in 3D CAD modeling, allowing us to leverage on industry standards file formats and build on open-source tools. We also focus on establishing full automation and coordination between multifunctional printhead processes and stage motion as well as develop control systems that are printer independent and could be adapted to other hardware implementations. This work also aims on developing a generalized description language and design approach to enable community development of printed tissues.

Chapter 2: Design and Implementation

In this work, we have developed a novel 3D bioprinting system that integrates tissue design software and microfluidics to create customizable 3D fiber-based heterogeneous tissue constructs [61], [62]. Following the printing process, cell-laden structures fabricated using our bioprinting system must undergo a process of cell culturing so that the cells differentiate appropriately. The following chapter will present an overview of the hardware components as well a detailed explanation of the design and implementation of our 3D bioprinting software toolchain

2.1 3D Bioprinter Hardware

The 3D bioprinting system developed in this work is mainly comprised of several hardware components: a microfluidic printhead, a pressure control system, a pneumatic solenoid switch bank, a vacuuming pump, a computer-controlled 3-axis positioning stage, and a customized printing substrate [61], [62]. The microfluidic printhead and the pneumatic solenoid switch bank were developed by Simon Beyer. A diagram of the developed bioprinting system is shown in Figure 8.

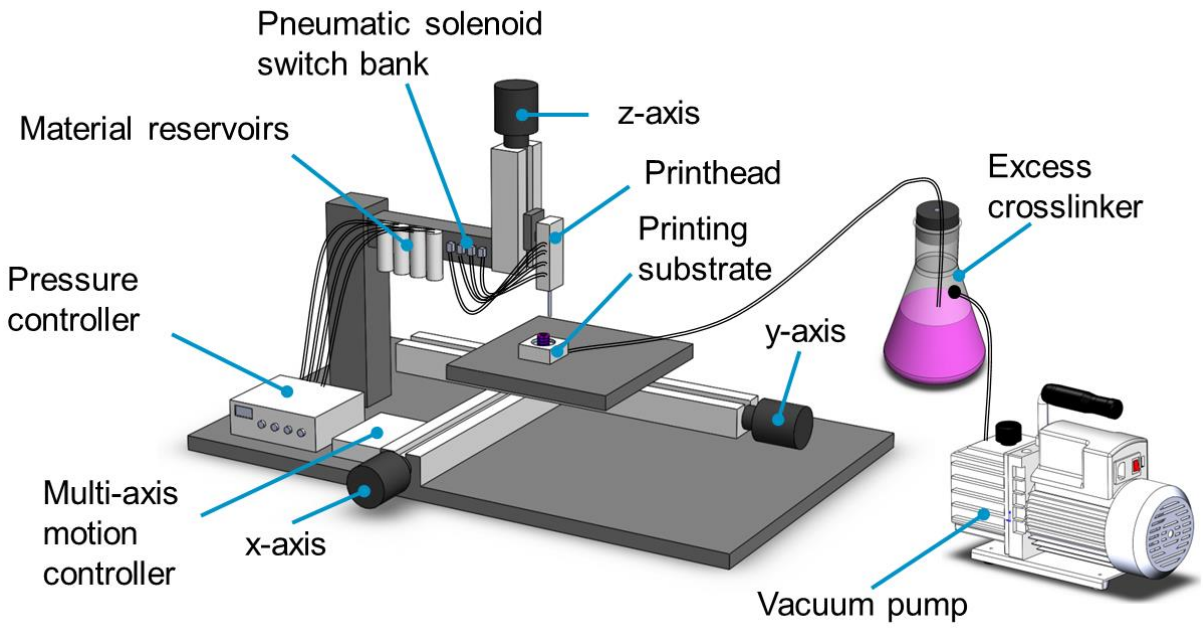


Figure 8: *Diagram of the 3D bioprinting system*

The printhead is comprised of a microfluidic chip capable of dispensing cell-laden hydrogel fibers. A stream of aqueous sodium alginate is coaxially focused by surrounding it with aqueous calcium chloride (crosslinker) within the printhead. In this way, sodium alginate is loaded into the printer as a liquid but dispensed as a gelled fiber (calcium alginate). Because of this, performing various on-chip microfluidic operations such as mixing and sequencing prior to gelling are possible. By changing the pressure ratio between the aqueous sodium alginate and the crosslinker, the fiber diameter could be varied from 70-300 µm. The on-chip flow rates are regulated using a pressure-driven flow controller (Fluigent MFCS-4C). To enable deposition of multiple materials, such as hydrogels laden with different types of cells, pneumatically controlled valves were

incorporated for each fluid channel in the printhead. A dedicated pneumatic solenoid switch was used to actuate the on-chip valve for each fluid channel. By changing the valves' configuration, the type of material being dispensed as well as the alginate gelation process could be controlled. The microfluidic printhead is fabricated from PDMS (poly-methylsiloxane) using a soft lithography process. The moulds used in the process are 3D printed using a commercial 3D printer (Objet 24).

Because the crosslinker exits the printhead orifice with the rapidly gelling alginate, it will pool and interfere with the already printed fibers. To overcome this problem, the current embodiment of the system must print materials on a porous substrate and vacuum must be applied to remove excess crosslinker during the printing process. Two different printing substrates with porous membranes were used. The first type of printing substrates used was standard well plate inserts (Corning® Costar® Transwell® cell culture inserts, 8.0 μm pore size). These are widely used and compatible with standard drug screening technology. The second type of printing substrates used was custom designed and 3D printed. The custom printing substrate contains a porous surface on which fibers are patterned and stacked. An outlet feed from the printing substrate is connected to a vacuuming system to remove the excess crosslinker. A schematic of the printhead and the customized printing substrate showing the removal of excess crosslinker is shown in Figure 9.

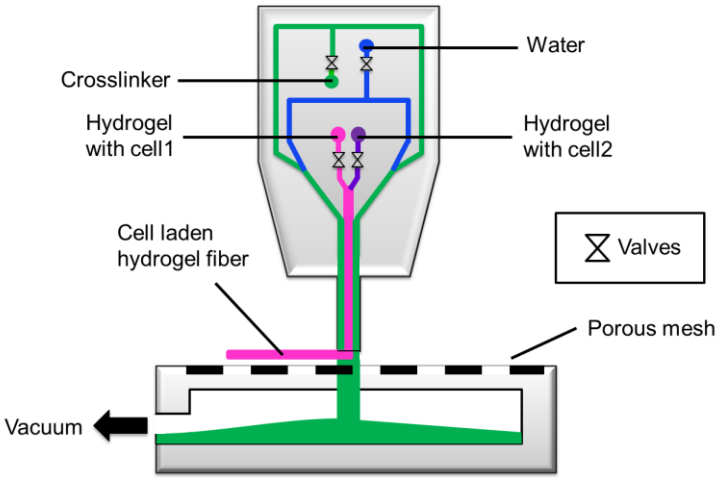


Figure 9: *Schematic of printhead and excess crosslinker removal mechanism*

To automate the process of patterning the gelled fibers deposited by the printhead, a computer controlled multi-axis positioning system was used. The positioning system is composed of X, Y, and Z axes. The printhead is attached to the Z axis and the printing substrate is secured onto both the X and Y axes. In this way, the printhead moves up in the *Z*-direction with each new layer according to the user-specified layer thickness. Within each layer, lateral patterning is achieved by coordinated movements in both the X and Y directions. The mechanical resolution of each of the three linear actuators is 1 µm. Three independent motor drivers supply each of the three stepper motors with the required pulse output signals that in turn provide the required motion to the different lead screws. For coordinated motion, a multi-axis motion controller was used (Galil DMC-2140). The Galil motion controller uses a specialized 32-bit Motorola 68331 series microcontroller [63]. The Galil hardware is programmed using an interpreted programming language called Digital Motion

Controller (DMC) code [64]. DMC code is run on a specialized Real Time Operating System (RTOS) that is loaded on the Galil hardware [64]. In addition to supporting embedded programming and running programs completely on the Galil controller, host-centric programming is also supported [64]. An application programming interface (API) can be used to send commands from the host computer to the controller [64]. For communication between the host computer and the motion controller, an Ethernet connection was used. Besides coordinated motion, this microcontroller serves as the 3D bioprinters' main processing unit and is used for the control and synchronization of external events, such as switching between different materials during a print job. Limits for each axis were detected using optical limit switches. A system level diagram is shown in Figure 10.

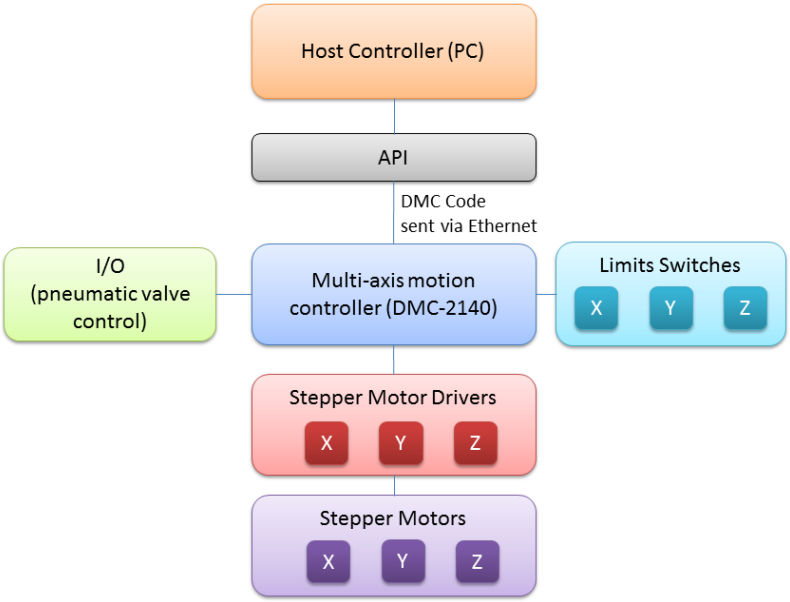


Figure 10: System level diagram

2.2 3D Bioprinting Software Toolchain

The developed software toolchain supports the design and 3D bioprinting of cell-laden hydrogel structures with intricate architectures, programmable compositions, and multiple printing properties such as fiber diameter, layer thickness, infill patterning, and porosity. The software toolchain is built around the idea of a component-based architectural design method. This design method is used in areas such as software engineering to develop large software-intensive systems as well as in aerospace engineering to design aircrafts composed of various different specialized subsystems [65], [66], [67], [68]. Bottom-up tissue engineering methods, such as the fiber-based technique used by our 3D bioprinter, are based around the same component-based design method. In the context of tissue printing, we have called this design approach the Tissue Building Block Design (TBBD) method. Using the TBBD method, complex tissues could be assembled from a set of simpler building blocks as shown in Figure 11. These building blocks can be placed in 3D either vertically or laterally. Then the arranged building blocks are printed layer-by-layer to realize the desired 3D structure.

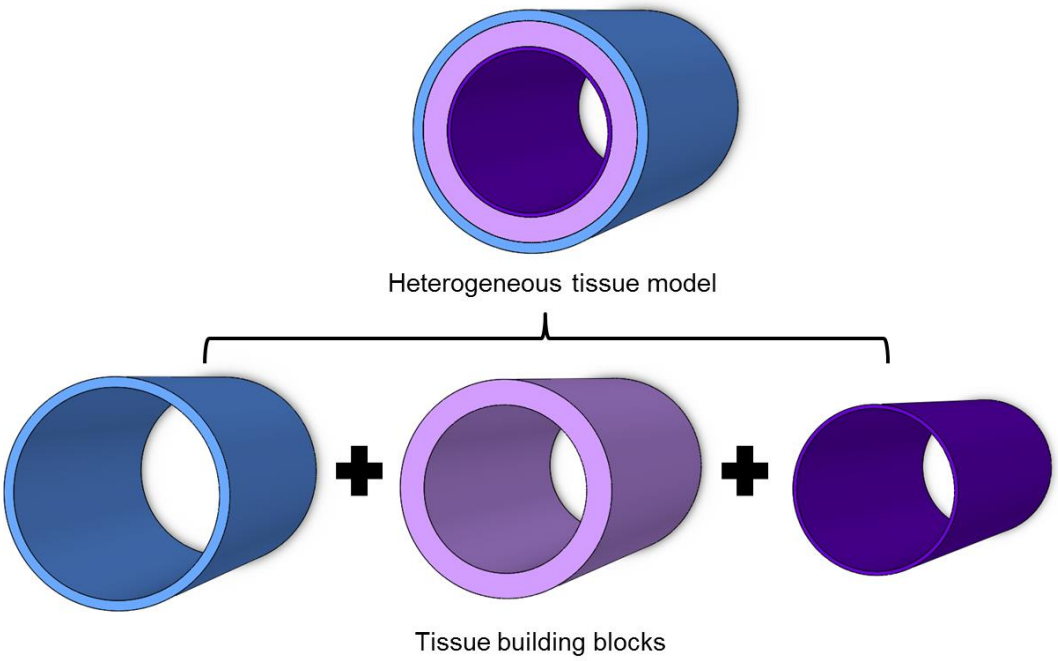


Figure 11: *Assembling of tissue building blocks*

A diagram of our developed software toolchain with the different software tools, file formats, and online platform is shown in Figure 12. The software tools include CAD Software, CAM Software, Tissue Designer, and Printer Control Software. The generated files include STL files, G-Code files, toolpath files, material files, and PTDL files. The workflow begins with designing different building blocks using 3D CAD software. Each building block is represented by one STL file. Then each STL file is sliced independently using CAM software according to specific CAM specifications and exported as a G-Code file. After that, the set of G-Code files (set of sliced building blocks) are merged into a single toolpath file and a single material file using Tissue Designer. Finally, the

generated toolpath and material files are interpreted by the Printer Control Software and commands are sent to the 3D bioprinter in order to execute the specified print job. An example showing the different steps involved in 3D printing a heterogeneous tissue structure designed using our bioprinting software toolchain is shown in Figure 13. To encourage the collaborative development and sharing of tissue design files amongst users, Tissue Designer also supports a Printed Tissue Description Language (PTDL) used to describe the designed tissue structure. An online platform called OpenVitro was initiated to facilitate the sharing of tissue design files, including PTDL files.

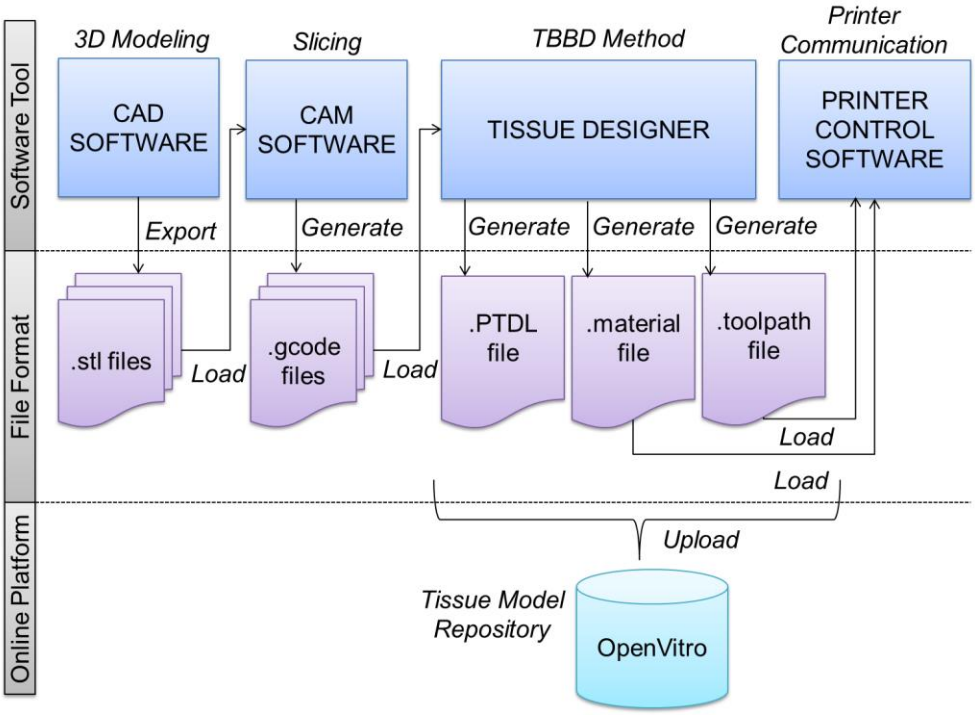


Figure 12: *Software tools, file formats, and online platforms used in our 3D bioprinting software toolchain*

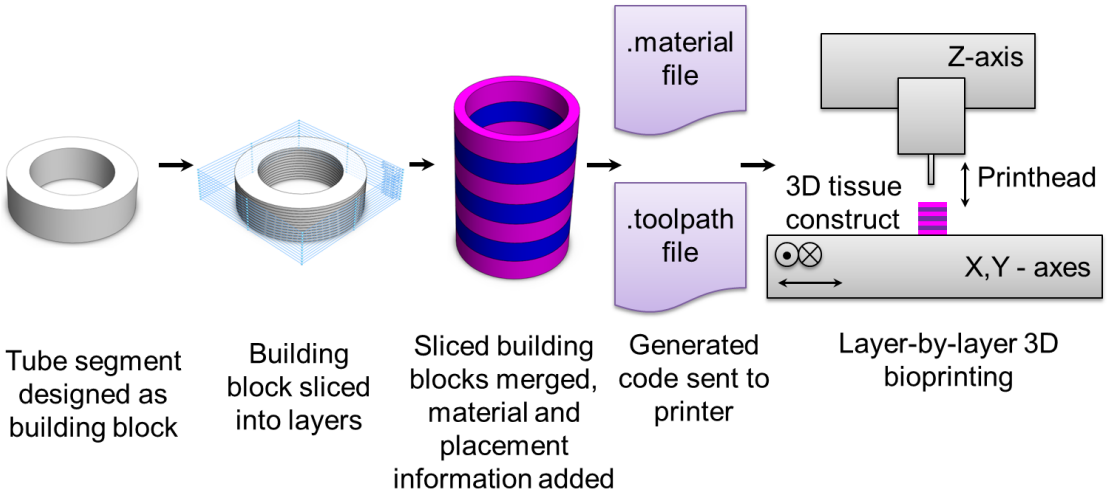


Figure 13: *Steps involved in the 3D bioprinting software toolchain*

The next sections will describe the details behind the different components of our 3D bioprinting software toolchain. In the first section, we will discuss the process of designing structures using the TBBD method. In the second section, we will discuss the used open-source slicer and the relevant CAM specifications. Then in the third section, we will discuss the novel algorithm used to merge individual tissue building blocks. In the fourth section, we will explain the details behind Tissue Designer, PTDL, and OpenVitro. In the last section, we will discuss printer control, synchronization, and communication.

2.2.1 Tissue Building Block Design Method

The TBBD method aims to allow for the design of a heterogeneous tissue model composed of a set of tissue building blocks. Each building block could be given different material assignments, CAM specifications, and other parameters that relate building blocks to one another, such as 3D orientation and print priorities. Tissue building blocks can be designed using any standard 3D CAD software that supports the generation of STL files. SolidWorks, one of the most popular 3D CAD programs [69], was used for the design of most building blocks in this work. To create a heterogeneous tissue structure, an individual building block must first be designed and exported as a STL file for each region that requires a different surface geometry, material type, or set of CAM specifications. Using SolidWorks, an assembly file representing the surface geometry of the entire tissue structure may be designed and the individual parts representing the different building blocks may be exported into separate STL files [69]. As previously mentioned, since STL files only capture surface geometries, CAM specifications and material assignments are assigned to each building block at subsequent stages of the toolchain.

For example, the heterogeneous structure in Figure 14 requires separate STL files describing the surface geometry of each building block. Using the TBBD method, the heterogeneous structure is created by merging the two different building blocks. Each building block will be individually sliced according to a set of CAM specifications

(Section 2.2.2) and then materials and other properties will be assigned (Section 2.2.3) in later stages of the 3D bioprinting software toolchain.

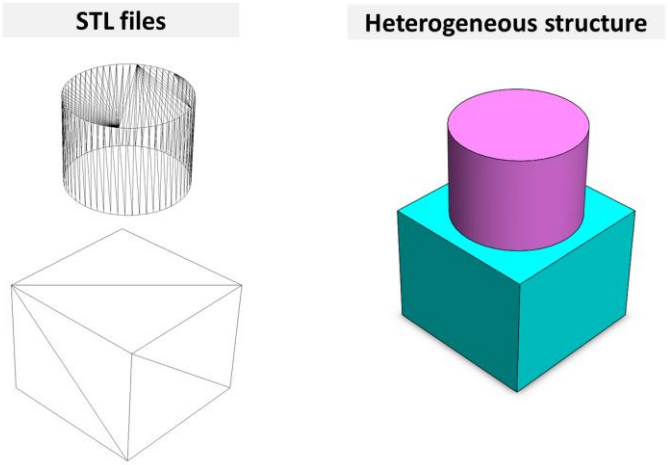


Figure 14: *Building blocks with different surface geometries*

In the case where two building blocks of a tissue structure have the same surface geometries, then only one STL file is needed. An example of this case is shown in Figure 15. The desired heterogeneous tissue structure is composed of two stacked tube segments, with the material type being the only difference between them.

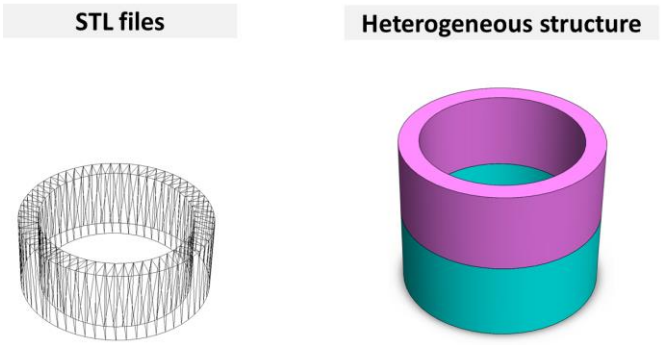


Figure 15: *Building blocks with identical surface geometries*

2.2.2 Tissue Building Block Slicing

As previously explained in section 1.3.2, the 3D printing process is a layered manufacturing technique that relies on slicing a 3D CAD model into a set of stacked horizontal layers and the generation of the necessary toolpaths to fill them. In order to generate a set of slices from a building block, an open-source slicing tool (Slic3r Version 0.9.8) was used. The input of the slicer software is a building block STL file. Usually, slicer tools allow the user to rotate the imported STL file to the desired orientation. Moreover, most open-source slicer programs are packaged with various functions and settings specifically optimized for the commonly used FFF printing method. For the purpose of our 3D bioprinting software toolchain, the open-source slicing tool is only used to generate the slices and the associated toolpaths needed to fill them. This information is described using G-Code. In this way, any other G-Code generators such as SkeinForge, Repsnapper, and Cura may be used instead of Slic3r. In other words, our 3D bioprinting software toolchain is not reliant on any one particular slicer software. Additionally, different slicer programs could be used to individually slice each building block, giving the user added design flexibility. The CAM specifications that are of most interest to us during the slicing process and their corresponding descriptions are summarized in Table 2.

Table 2: *CAM specifications for the slicing process*

CAM Specification	Description
Layer thickness (mm)	The vertical distance between each consecutive layer
Number of perimeters	The number of outlines or contours per layer
Fill density (%)	The percentage of the area within a perimeter that is filled with material
Fill pattern	The specific geometry that is used to fill the area within a perimeter
Extrusion width (mm)	The width of the deposited fiber within a layer

As an example, an STL file describing the surface geometry of a cylinder with a diameter and height of 10 mm is sliced (Figure 16). The layer thickness is set to 100 µm, resulting in the generation of a sliced building block that consists of 100 slice elements. The number of perimeters is set to 2. The fill density is set to 40%, which corresponds to 60% porosity. The extrusion width is set to 100 µm. A diagram showing the 50^{th} slice of the cylinder with different fill patterns is shown in Figures 16. Different fill patterns could be used to achieve intricate internal tissue architectures that could affect biological response [22].

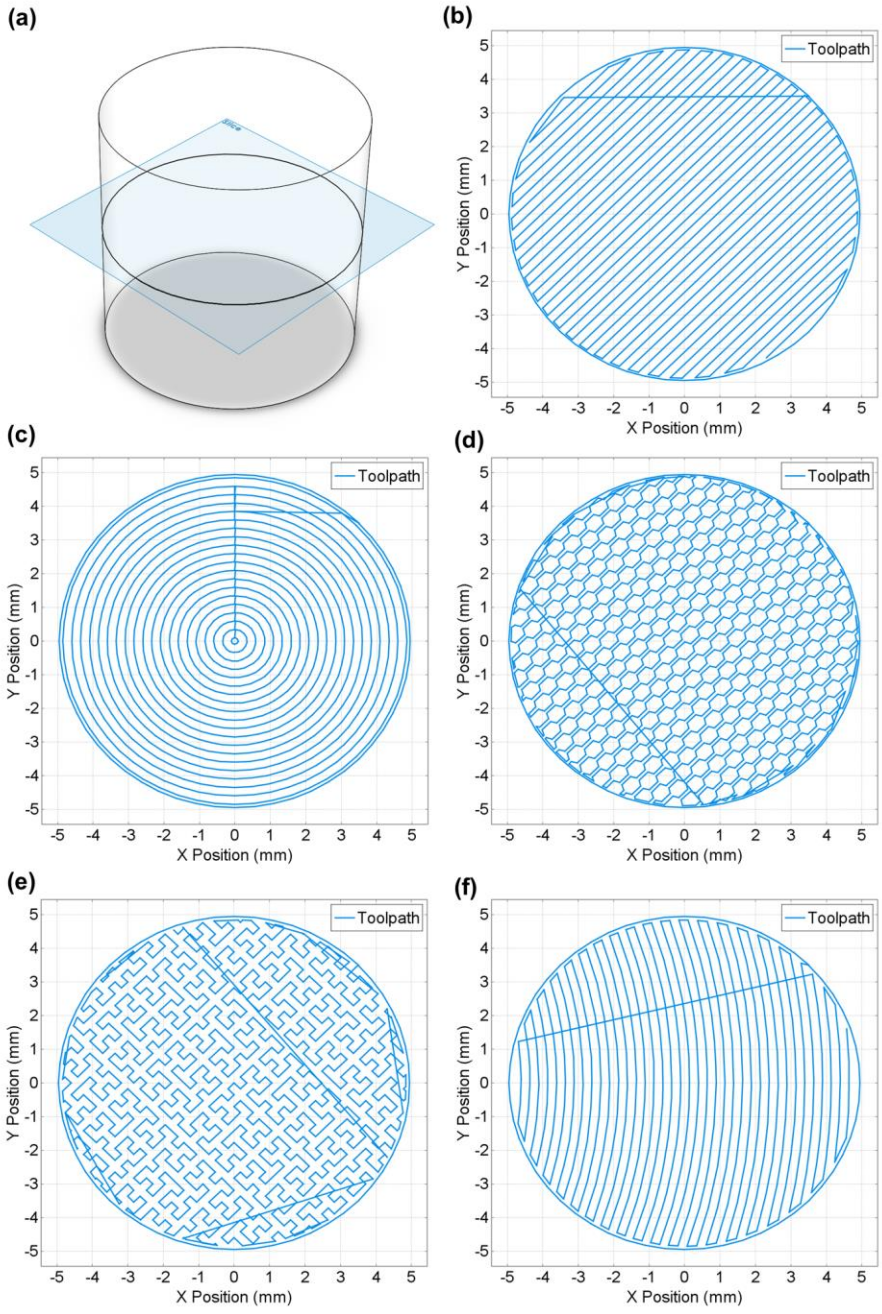


Figure 16: *(a)Slice of a cylinder (b) rectilinear fill pattern (c) concentric fill pattern (d) honeycomb fill pattern (e)hilbertcurve fill pattern (f) archimedeanchords fill pattern*

The G-Code file generated from the slicing process consists of a series of G1 commands. As previously mentioned, a G1 command denotes linear interpolation between the current absolute position and the specified absolute position. In other words, the generated G-Code program describes a toolpath that consists of a series of straight line movements. A typical generated G-Code file consists of a structured list of three different types of G-Code block patterns shown in Table 3. We define a Type I G-Code block as a block that describes a straight line movement within a layer while depositing material. A Type II G-Code block describes a straight-line movement within a layer without depositing material. A Type III G-Code block describes a layer change.

Table 3: Different types of defined G-Code blocks

G-Code Block Type	Code	Description
Type I	G1 XNumber YNumber FNumber ENumber	Move from the current XY position to the specified position (XNumber, YNumber) at the specified feed rate (FNumber) while extruding the specified length of filament (ENumber)
Type II	G1 XNumber YNumber FNumber	Move from the current XY position to the specified position (XNumber, YNumber) at the specified feed rate (FNumber) without extruding
Type III	G1 ZNumber FNumber	Move from the current Z position to the specified Z position (ZNumber) at the specified feed rate (FNumber)

To follow an incremental tissue development approach, a sliced building block could be printed. Then the design could be further modified based on observed structural and functional endpoints. Once the building block design is optimized and the printed structure is shown to exhibit the desired behavior, it could be integrated with previously optimized building blocks and printed as one heterogeneous structure. For a more rapid tissue development process or for experiments with endpoints that depend exclusively on the interaction between heterogeneous components, the sliced building blocks could be merged with other sliced building blocks and then the entire set could be printed at once.

2.2.3 Tissue Building Block Design Algorithm

The previous section discussed the second step of the 3D bioprinting software toolchain: slicing each building block according to specific CAM specifications. This section discusses the third step of the 3D bioprinting software toolchain: merging a set of sliced building blocks using the developed TBBD Algorithm. If building blocks are stacked on top of each other, such as the arrangement shown in Figure 15, a sequential printing approach whereby a given building block is completely printed before starting to print the subsequent building block may work. However, this approach will not work if, for example, the building blocks were arranged closely side-by-side. In this case, a collision would occur between the printhead and the first printed building block when it starts printing the second building block. In order to 3D print heterogeneous structures composed of multiple building blocks arranged side-by-side, such as concentric tubes,

we need to be able to merge slices from different sliced building blocks as the structure is built up layer-by-layer.

The input of the TBBD algorithm is a set of sliced building blocks represented by a set of G-Code files as well as a material number, printing priority, and XYZ parameters for each building block. Using our software, a multi-material structure could theoretically be composed of an infinite number of material types. However, in a practical sense, the number of materials is bounded by the number of material channels supported by the single-nozzle microfluidic printhead used in our bioprinting system. For this project, the microfluidic printhead architecture only supports a maximum of two materials. To address the issue of merging slices from different sliced building blocks at the same Z level, a priority-based scheduling scheme is used. A fixed, user-specified printing priority is assigned to each sliced building block during the design phase. Printing priorities are used in scheduling the order in which slices from different building blocks at the same *Z* level are printed. Higher priority slices are printed before lower priority slices. Algorithms could be developed in the future to automatically assign printing priorities based on specific design rules or application-specific requirements. In order to arrange a set of building blocks in 3D space, each building block is given an x-center position, y-center position, scaling factor, and z-offset. The x-center and y-center positions are used to orient the building block in the XY plane. For example, if x-center and y-center positions are equal to zero, the building block will be centered at the origin of the printing substrate. The scaling factor is used to scale the building block in the XY plane. The z-offset parameter is used to indicate the *z*-position of the first slice of a

building block. In other words, the z-offset parameter is used to stack building blocks on top of each other.

The TBBD algorithm is divided into three main modules: G-Code Interpreter, Slice Scheduler, and Output Code Generator (Figure 17). For fast processing and efficient memory usage, the algorithm was written using C++ [70]. The G-Code Interpreter analyses and parses through the input and stores it into different classes of dynamically created objects. The Slice Scheduler is responsible for scheduling the order in which slices are printed. The Output Code Generator receives slice objects from the Slice Scheduler and produces the toolpath and material code understood by the 3D bioprinter. Each of these modules will now be explained in more detail.

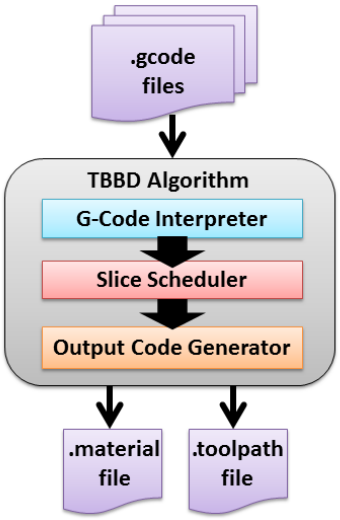


Figure 17: *TBBD Algorithm*

The G-Code Interpreter is developed using Flex and Bison (GNU implementations of Lex and Yacc) [71] . Flex and Bison are two powerful tools used to develop interpreters and compilers [71], [72]. Flex and Bison work together to find

patterns in structured input and produce a response based on the recognized pattern [71], [72]. This process is split up into two stages: lexical analysis, which is done using Flex, and parsing, which is done using Bison [71], [72]. Lexical analysis involves defining the input into significant units called tokens and describing each token using a regular expression [71], [72]. Our tokens are defined as the different G-Code words in a G-Code block. A grammar is specified in Bison and defines the relationships between a sequence of tokens and their meaning [71], [72]. Our defined grammar specifies the rules of how the defined tokens make up the different G-Code blocks. In runtime, Bison parses the G-Code using the defined grammar.

The G-Code Interpreter structures the input according to the class diagram shown in Figure 18. The G-Code Interpreter starts with creating a GCodeFile object for each input G-Code file (building block) and stores the associated material number, printing priority, x-center, y-center, scaling factor, and z-offset. The G-Code Interpreter then scans each G-Code file line-by-line and detects whenever a sequence of tokens matches the defined grammar. Whenever a valid G-Code block is detected, a function is called to store the numerical portions of the G-Code words and the type of the G-Code block in a GCodeBlock object. The created GCodeBlock object is added to an existing Slice object or a new Slice object. A new Slice object is created whenever we encounter a Type III G-Code block (see Table 3), which indicates a layer change. A data member called z_level in the Slice class is used to keep track of the absolute height of the slice. In other words, z_level is equal to the z-offset plus the numerical portion of the “Z” G-Code word of a Type III G-Code block; in this way, we can merge building blocks that

are sliced with different layer thicknesses. This process continues until the end of the valid G-Code file. Storing the G-Code in a structured way, as opposed to just a sequence of instructions, makes our software powerful, flexible, and scalable. A basic flowchart showing how one valid input G-Code file and the associated parameters are processed and stored using the G-Code Interpreter is shown in Figure 19.

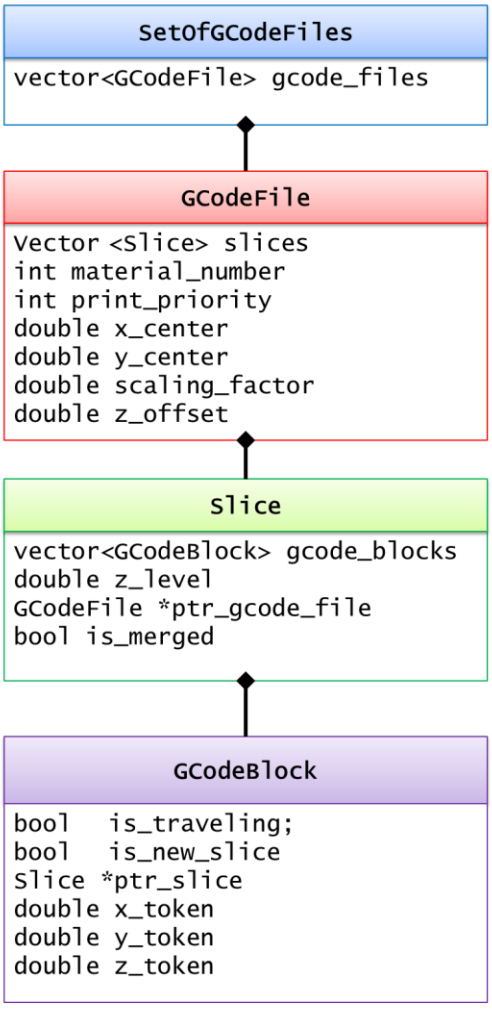


Figure 18: *Class diagram showing composition relationship*

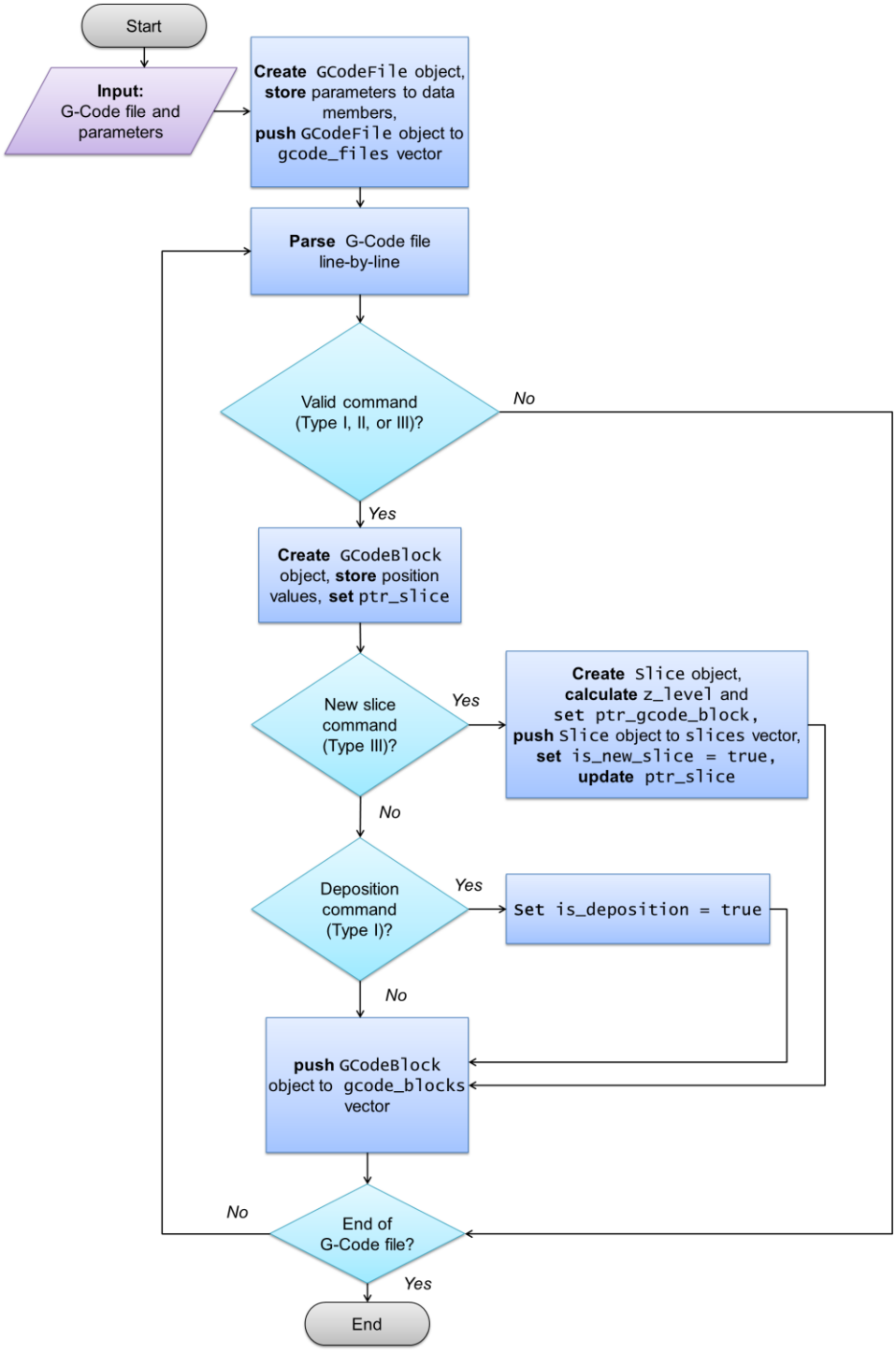


Figure 19: *Flowchart of G-Code interpreter module*

The Slice Scheduler is responsible for determining which Slice object is sent to the Output Code Generator to be printed next. Since the slices are correctly sorted within each building block, the z_level and print_priority of each Slice object at the front of the vectors are compared. Among these compared Slice objects, the one with the lowest z_level and highest print_priority is scheduled to be printed next. If the z_level of the current slice object is equal to the z_level of the previous slice object, then the two slices are merged. In other words, the printhead will not move up in the Z direction and both toolpaths are combined. This is done by setting the is_merged flag to true. This flag informs the Output Code Generator to ignore any Type III (layer change) G-Code blocks for merged slices. As an example, a graph showing the toolpath of two merged slices, each slice coming from separate building blocks is shown in Figure 20. A flowchart of the Slice Scheduler is shown in Figure 21.

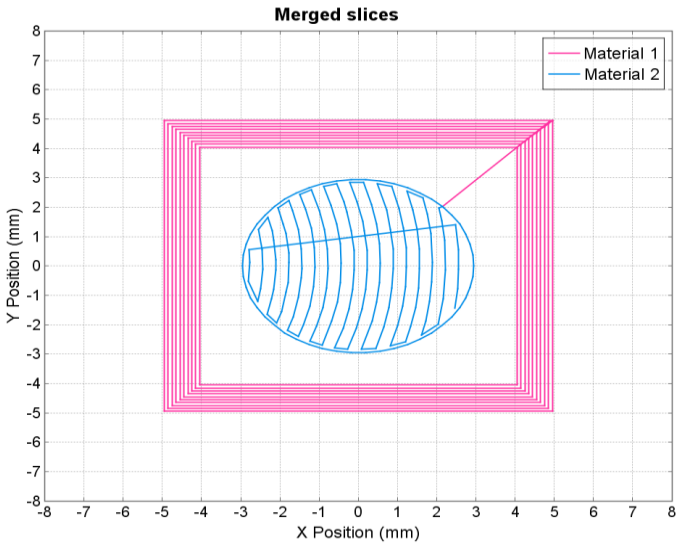


Figure 20: *Merged slices*

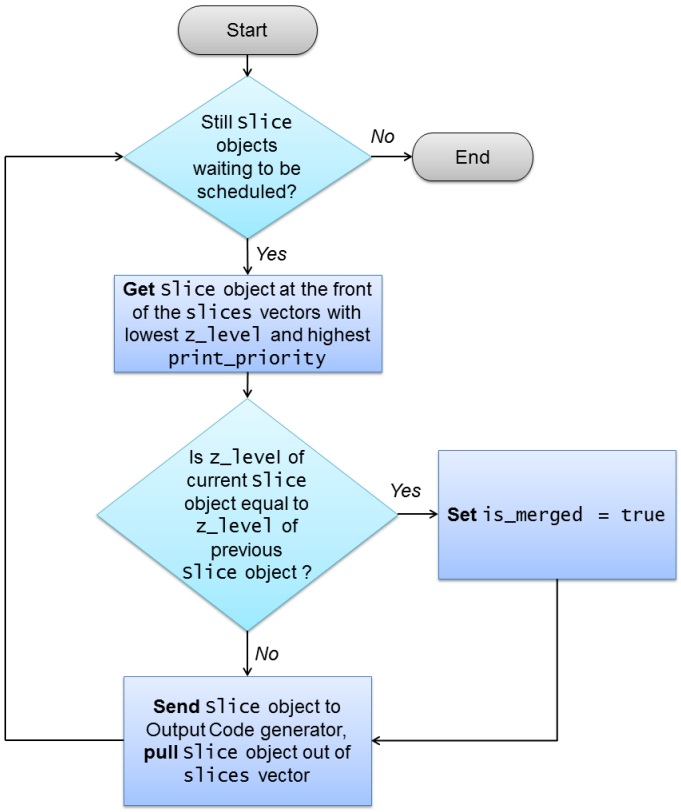


Figure 21: *Flowchart of Slice Scheduler*

The Output Code Generator is mainly responsible for producing two files used to drive the 3D bioprinter: a toolpath file and a material file. As previously mentioned, due to the lack of a universal G-Code adopted by motion control manufactures, G-Code interpreting or post-processing is a necessary step [32], [33]. Furthermore, some motion controllers don't understand any form of G-Code altogether. This is the case with the motion controller used in our 3D bioprinting system. The motion controller used only understands DMC code. A "G1" G-Code command is equivalent to a "LI" DMC command. Furthermore, the G-Code generated by the slicer application is in absolute coordinates, but the Galil controller requires relative coordinates. Additionally, the

generated G-Code is in millimetres, whereas the DMC controller requires encoder counts. The DMC code structure for sending a sequence of linear interpolation commands is shown in Table 4.

Table 4: DMC code structure for a linear interpolation sequence

DMC Instruction	Description
LM ABC	Specify the number of axes
LI 3000, 2000,1000	Move 3000 counts in X, move 2000 counts in Y, and 1000 counts in Z from the current XYZ position
. . .	. . .
LE	End of sequence

The generated toolpath file contains DMC code that specifies the list of linear interpolated movements required to print the designed 3D model. The Output Code Generator waits until it receives a Slice object from the Slice Scheduler. Once a Slice object is received, the Output Code Generator loops through all the GCodeBlock objects in the gcode_blocks vector (which is stored in the Slice object) and translates G-Code to DMC. The scaling_factor is also applied to the XY position movements at this stage. If the is_merged flag value is true, the Z movement of the GCodeBlock object is ignored. If the is_traveling flag value is true, it is possible to invoke a function to stop depositing material, but our printhead currently does not support this.

The material file consists of two space delimited columns: the first column specifies the material type and the second column specifies the total fiber length (µm) that has to be printed (from the start of the print job) before switching to the corresponding material type. To compute each of these values, the total fiber length is updated each time the Output Code Generator processes a GCodeBlock object using the following equation,

$$L_{fiber} = \sum_{i=1}^{n} \sqrt{x\_displacement_i{}^2 + y\_displacement_i{}^2 + z\_displacement_i{}^2}. \qquad (1)$$

When the Output Code Generator detects a change in material type, a new row is inserted in the material file containing the material number and the current fiber length. The structure of a material file is shown in Figure 22. For example, when the total printed fiber length is equal to 438483 µm, material 1 will be loaded and printed; when the total printed fiber length is equal to 589769 µm, material 2 will be loaded and printed; when the total printed fiber length is equal to 666751 µm, material 3 will be loaded and printed; when the total printed fiber length is equal to 779290 µm, material 4 will be loaded and printed. A flowchart of the Output Code Generator is shown in Figure 23.

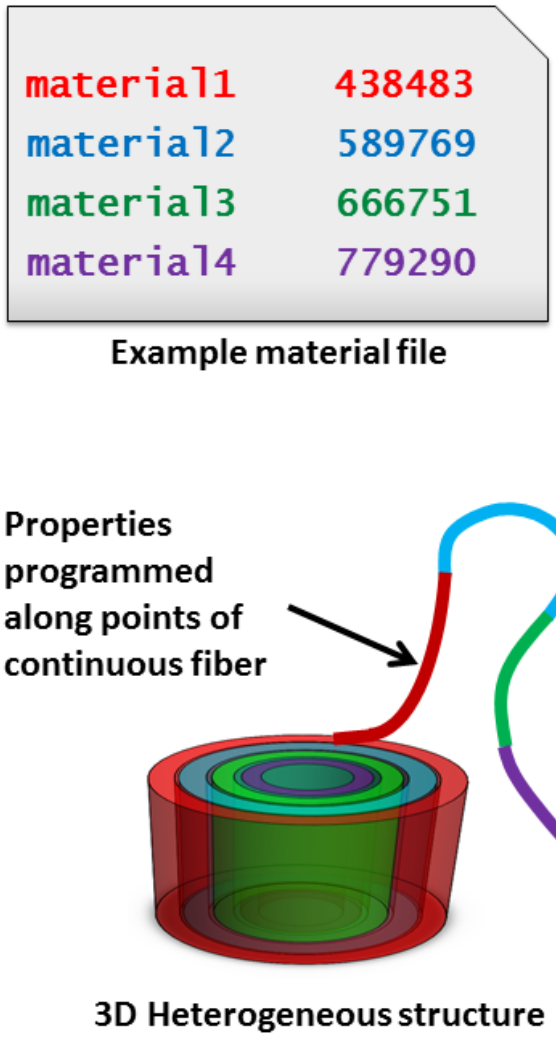


Figure 22: *Structure of material file*

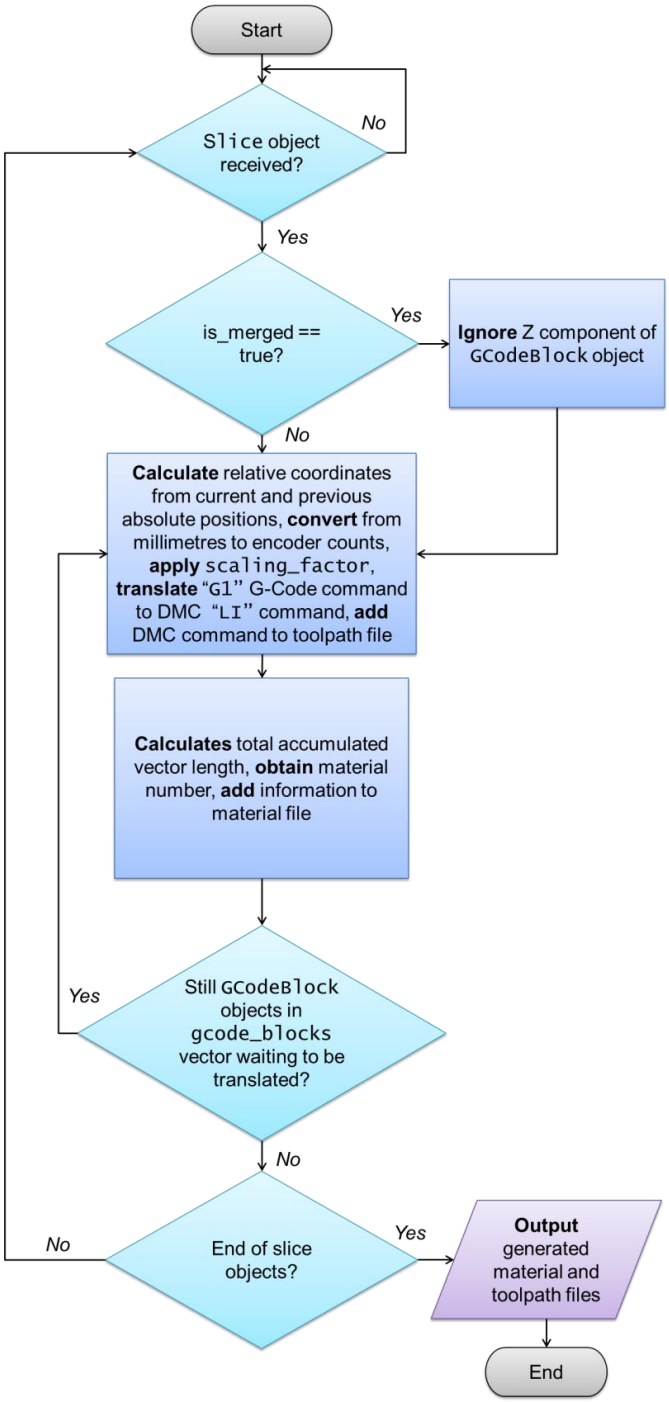


Figure 23: *Flowchart of output code generator module*

The calculated total fiber length is also used to estimate the build time for a given print job. The build time is approximately equal to the total fiber length divided by the print speed,

$$Build\ time\ \approx \frac{Total\ fiber\ length}{Speed}. \quad (2)$$

The estimated build time allows the user to better evaluate costs before they are incurred and better plan printing experiments.

In addition to generating the material and toolpath files, a folder containing the point clouds of each layer is outputted. The point clouds are graphed and used to analyze the generated toolpaths, layer-by-layer, before printing. This could be important as users could detect problems before starting the printing process, which could potentially save time and material costs. Examples of issues that may be detected by visualizing the point clouds include overlapping toolpaths as well as trailing fibers caused by Type II G-Code blocks.

With minor changes, a new Output Code Generator capable of generating code understood by different motion controllers could be easily developed. This is the power behind using the Flex and Bison. For example, recently we have developed a second 3D bioprinting system that uses a different motion controller model. Slight modifications to the output code generator module were made, and a new version of our software was released.

2.2.4 Tissue Designer, PTDL, and OpenVitro

In addition to developing a Linux-based console interface, an intuitive graphical user interface (GUI) called Tissue Designer was developed. Tissue Designer allows users with no programming experience to assemble building blocks into heterogeneous tissue structures. This was an important requirement as initial users of our 3D bioprinting software toolchain include biologists with no programming backgrounds. Both the console interface and Tissue Designer allow users to input a set of sliced building blocks and the associated parameters, run the TBBD algorithm, and generate the output code to drive the 3D bioprinter. Tissue Designer was written using Visual C# and is built around the developed core TBBD algorithm (section 2.2.3).

The Tissue Designer interface is shown in Figure 24. The software allows users to “Drag and Drop” or “Add” a set of G-Code files (sliced building blocks) and specify the associated material numbers (MTRL), printing priorities, placement parameters (X, Y, and Z), and scaling factors (SF). The printing priority of each sliced building block is assigned by using the “Up” and “Down” buttons; the top row is assigned the highest priority and the bottom row is assigned the lowest priority. A specific building block can be removed from the “PTDL Palette” by using the “Delete” button. The PTDL Palette can be entirely cleared by clicking the “Delete All” button. Before merging the building blocks (i.e., run the TBBD algorithm), the specific printer model must be specified. Currently, our printer supports two in-house developed 3D bioprinting systems that we have called Biogen 1 and Biogen 2. After the merging process is successfully

completed using the core TBBD algorithm, the generated toolpath and material files can be exported so that they can be fed into the printer control software.

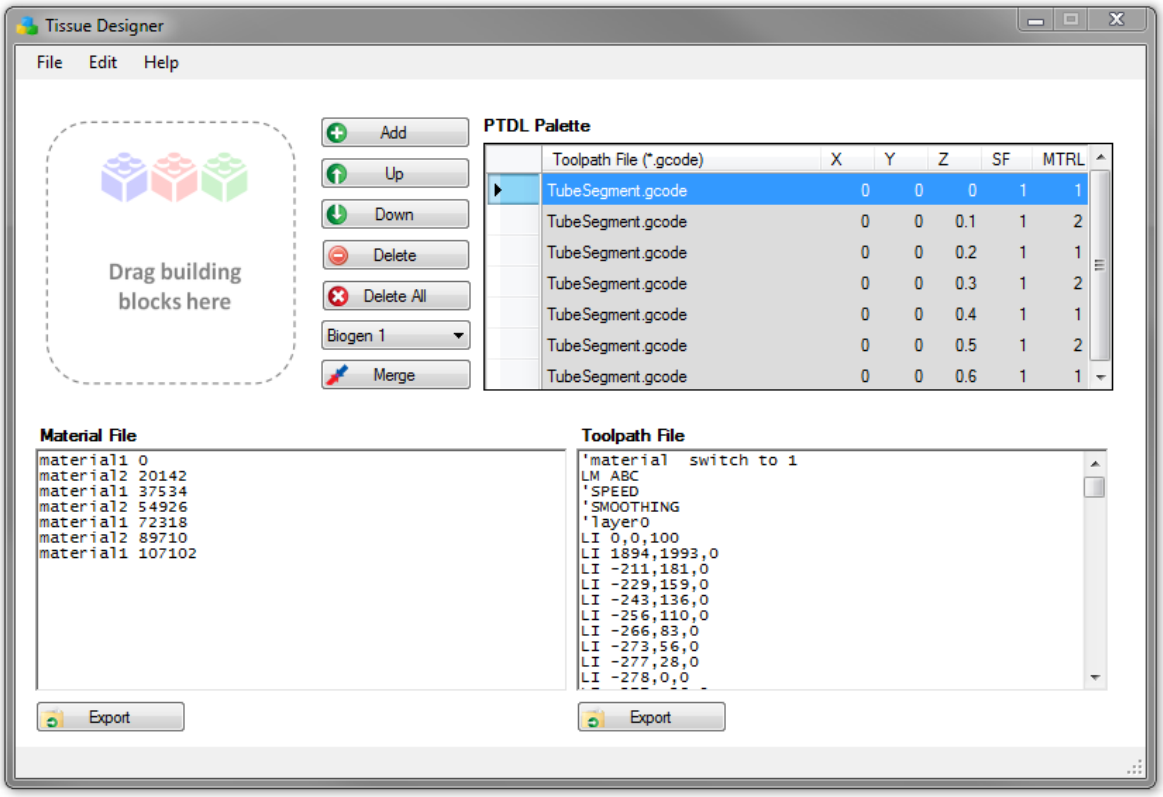


Figure 24: *Tissue Designer graphical user interface*

One important feature of Tissue Designer is that it allows users to export and import the assembled heterogeneous structures using our developed PTDL format. PTDL is an Extensible Markup Language (XML) based format for describing printed tissue structures. XML is an open standard that is managed by the World Wide Web Consortium (W3C) [73]. XML is an ASCII file that contains structured data organized

into a set of hierarchical elements. Because of its wide acceptance, there exist many open-source tools for performing operations on XML files such as editing, viewing, parsing, compressing, and encrypting.

The purpose of PTDL is to facilitate the exchange and reuse of printed tissue descriptions using a common language. This would reduce the time spent redefining previously described tissue components and encourage the design of more complex structures. Both the export and import PTDL features can be accessed from the file menu in Tissue Designer.

The PTDL syntax is built to support the TBBD method. The basic PTDL format is composed of several elements shown in Table 5. The first line of the PTDL file describes the XML version and the encoding used. The rest of the elements in the PTDL file are enclosed within a parent root element, <ptdl> and </ptdl>. Within the parent element, building blocks are specified as child elements, sorted by decreasing printing priority. Within each building block child element, the building block properties are specified as subchild elements. An example PTDL file containing two building block elements is shown in Figure 25.

Table 5: *PTDL elements*

Element	Description
<ptdl>	Parent root PTDL element
<block>	Defines a new building block element
<gcode>	Specifies a reference to a g-code file
<x>	Specifies the x-center position of the building block
<y>	Specifies the y-center position of the building block
<z>	Specifies the z-offset position of the building block
<scale>	Specifies the XY scaling factor of the building block
<material>	Specifies the material number of the building block

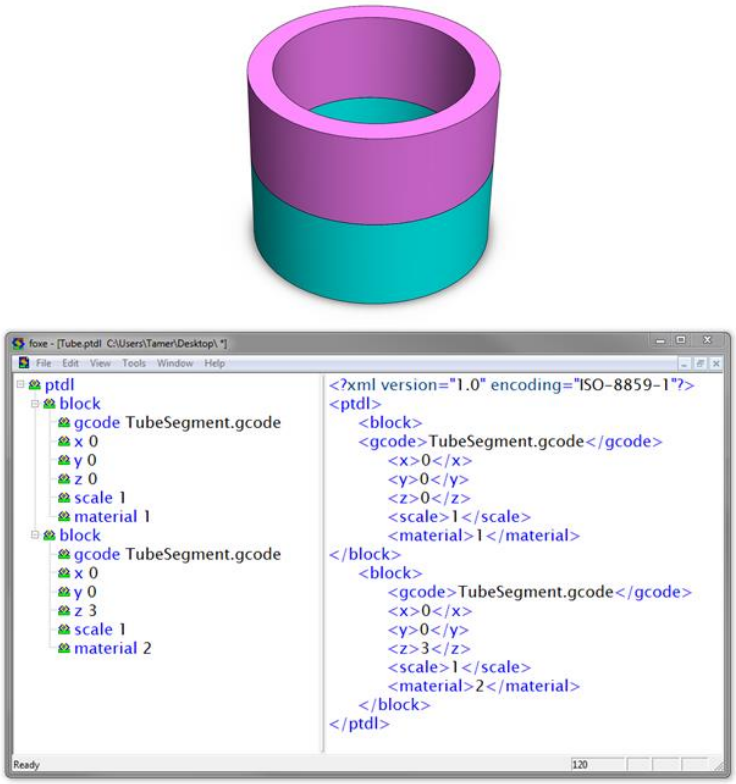


Figure 25: *PTDL file describing a heterogeneous tube*

To provide an online platform for exchanging tissue models, OpenVitro was initiated (Figure 26). OpenVitro is a website for sharing PTDL and other tissue design files, such as STL files, G-Code files, toolpath files, and material files. The long-term goal of OpenVitro is to facilitate the collaboration amongst biologists and tissue engineers. Through OpenVitro, PTDL and different elements of the 3D bioprinting software toolchain could be further developed based on elicited user-requirements and feedback.

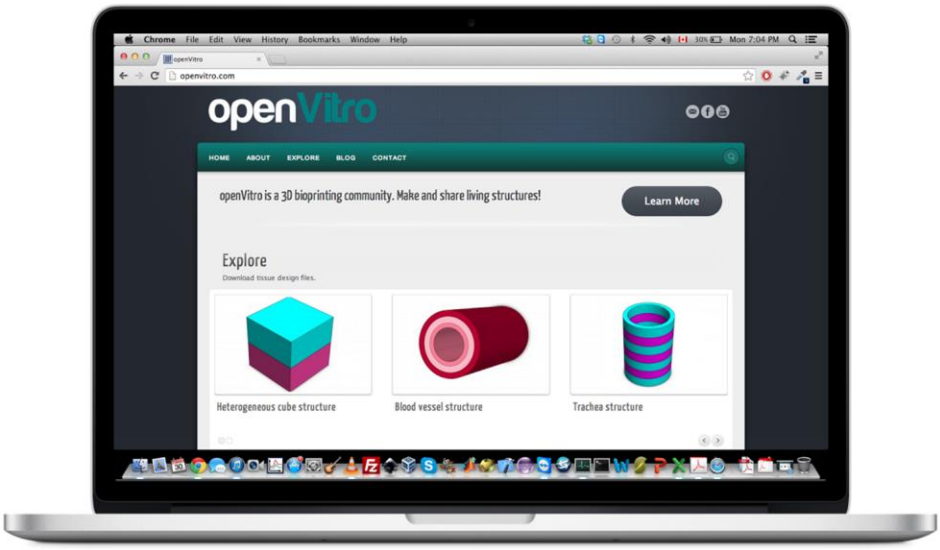


Figure 26: *OpenVitro online tissue model sharing*

2.2.5 Printer Control, Synchronization and Communication

Control over on-chip behavior and synchronization between printhead function and stage motion is essential for printer automation. To do this, the microcontroller is used to generate output signals controlling on-chip valves' configurations based on specific events. In this way, the printhead could be programmatically manipulated. Examples of

current printhead functions include chip initialization and cleanup, the initiation and halting of fiber generation, and material switching. Future printhead functions could include material mixing and concentration gradient generators.

To initialize the printhead before printing, a function was developed that consists of a timed-sequence of on-chip valve configurations. The main purpose of the initialization procedure is to fill the different on-chip channels with the appropriate materials and confirm proper chip function prior to printing. The pressure values for all the material inputs were maxed to 75 mbar. The entire procedure is listed in Table 6. First, all the material valves (water, crosslinker and both hydrogel pre-cursors) are opened for five seconds. This is done to get rid of bubbles by allowing liquid flow through all the channels. Then the water valve is closed and the remaining valves are kept open for two seconds to confirm that a fiber is being generated. Then the exiting channel is flushed from any residue by closing all the valves except for water for fifteen seconds. Finally, all the valves are closed. After the completion of the initialization/cleanup procedure, the printhead should be ready for printing

Table 6: *On-chip valve configurations for initialization/cleanup procedure*

#	Duration (Seconds)	Water	Crosslinker	Alginate 1	Alginate 2
1	5	**OPEN**	**OPEN**	**OPEN**	**OPEN**
2	2	**CLOSED**	**OPEN**	**OPEN**	**OPEN**
3	15	**OPEN**	**CLOSED**	**CLOSED**	**CLOSED**
4	-	**CLOSED**	**CLOSED**	**CLOSED**	**CLOSED**

To keep track of the amount of fiber deposited during a motion sequence, the total distance traveled from the start of the toolpath is continuously stored and updated in real-time on the microcontroller's on-board memory. By monitoring this value and accounting for delays and execution times associated with on-chip procedures, unique printhead operations could be executed at controlled points along a continuously deposited fiber. Though our printhead only supports material switching, as the printhead is further developed and other features such as concentration and temperature control, cell counting and mixing are integrated into the hardware, continuous fibers with more advanced and spatially-controlled properties could be printed.

As mentioned above, printhead tasks should be executed *a priori* so that properties could be applied in a controlled way along different points of a continuously printed fiber. When printing multi-material structures, the printhead needs to ensure that the specified material is being deposited at the correct time. Because the process of switching materials occurs within the chip a distance away from the tip of the nozzle, there is an unavoidable delay associated with material switching that should be accounted for. In other words, material switching needs to be triggered on-chip prior to a material switch point within a toolpath to allow for the previous material within the channel to exit the nozzle and the new material to fill the channel. As long as the fiber exits the printhead nozzle at a rate equal to the stage speed, the distance (channel length) it takes within the printhead to change materials should be constant. If the fiber exits the nozzle at a rate that is faster than the stage speed, the fiber will coil within the channel, causing the effective channel length to be greater than the actual length.

Besides affecting the channel length, coiling within the channel may also lead to eventual chip clogging. If the fiber exits the nozzle at a rate that is slower than the stage speed, the fiber will be stretched. This will also affect the channel length and may also lead to other problems such fiber breakage. These three fiber exit velocity scenarios are summarized in Figure 27.

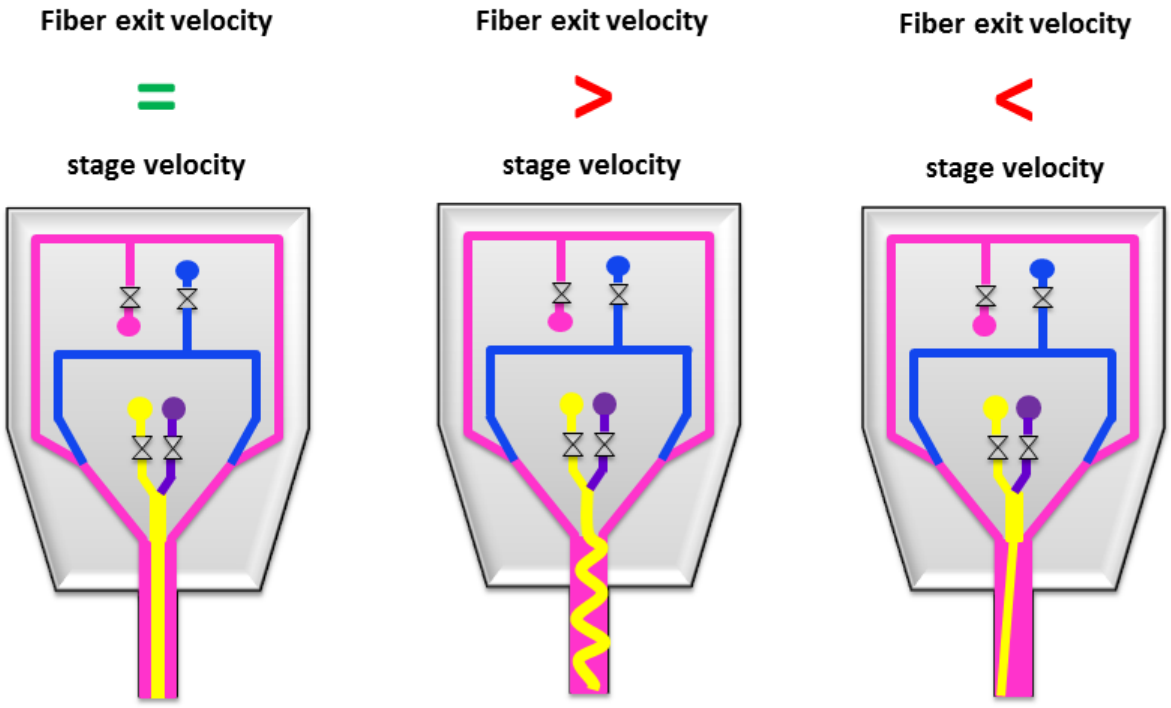


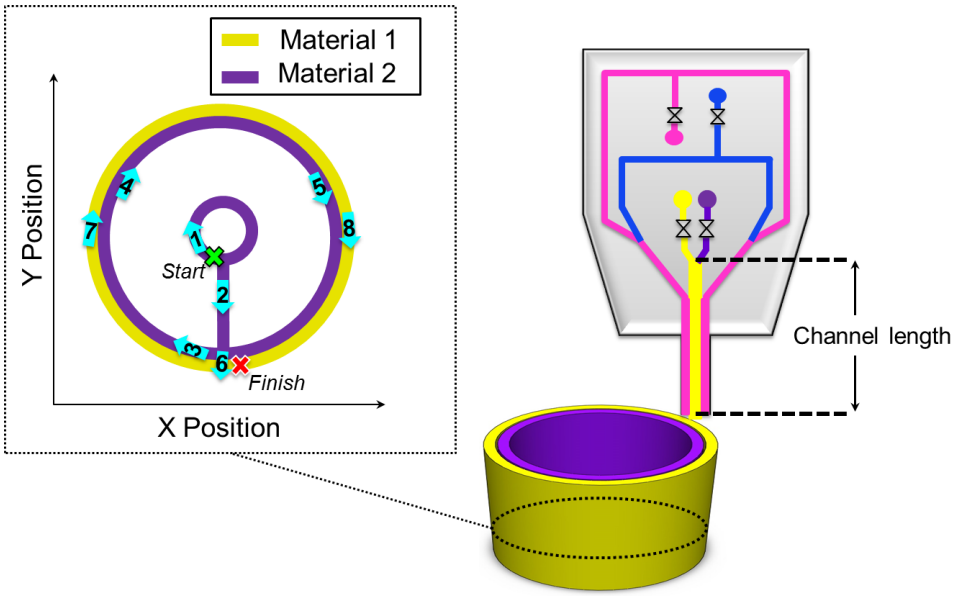
Figure 27: *Fiber exit velocity versus stage velocity*

As mentioned in section 2.2.3, the TBBD algorithm generates a material file with several switch points. These switch points correspond to the length of the fiber to be deposited and the material type for each building block. While only two commands are currently supported (material1 and material2), additional commands could be defined to

accommodate for tasks other than material switching. By comparing the distance traveled (which is continuously updated on the microcontroller's on-board memory) and comparing it with the defined switch points, unique chip conditions could be invoked at the beginning of a building block's toolpath. In this way, appropriate properties could be applied to each building block. The ability to programmatically apply multiple properties to different regions of a 3D printed tissue pre-cursor could help ensure that the post-printing cell culturing process enables cells to differentiate appropriately and exhibit a biological function.

To explain how material switching is coordinated with the motion toolpath using the approach mentioned above, an example is presented in Figure 28. The 3D structure shown in the example is a multi-material coaxial tube. Several key points are labeled along the toolpath and the corresponding valve configurations are summarized. To enable fiber gelation throughout the entire toolpath, the water valve remains closed while the crosslinker valve stays open. At point 1, material 2 valve is opened while material 1 valve is closed, allowing for the deposition of purple fiber. Usually, an initialization structure is printed some distance away to ensure that the channel is primed with the appropriate material and that a uniform fiber is being deposited by the first point of the desired structure's toolpath. By point 2, the initialization structure should have printed successfully and the toolpath starts moving towards the first point of the structure. The valve configurations remain unchanged from point 1 to point 5. At point 5, the remaining arc length for the inner purple circle is equal to the channel length. Therefore, material 2 valve is opened and material 1 valve is closed. By point 6, all

material 2 should have exited the channel and material 1 is deposited. These valve configurations remain unchanged from point 6 to finish. After the toolpath is completed, all valves are closed.



#	Water	Crosslinker	Material 1	Material 2
1	CLOSED	OPEN	CLOSED	OPEN
2	CLOSED	OPEN	CLOSED	OPEN
3	CLOSED	OPEN	CLOSED	OPEN
4	CLOSED	OPEN	CLOSED	OPEN
5	CLOSED	OPEN	OPEN	CLOSED
6	CLOSED	OPEN	OPEN	CLOSED
7	CLOSED	OPEN	OPEN	CLOSED
8	CLOSED	OPEN	OPEN	CLOSED

Figure 28: *Example of synchronization between material switching and toolpath*

Printer control software was developed to facilitate control over printer function. A GUI of the software is shown in Figure 29. The printer control software features can be divided into four categories: setup controls, motion controls, valve controls, and print

controls. Real-time control of the on-chip channel pressure values is handled by the pressure controller (Fluigent) software. Various buttons and inputs are implemented to provide users with useful print-related functions. A terminal console window is used to display output messages such as those relating to connections and disconnections of the bioprinter, system and software errors, print status, and on-chip valve configurations.

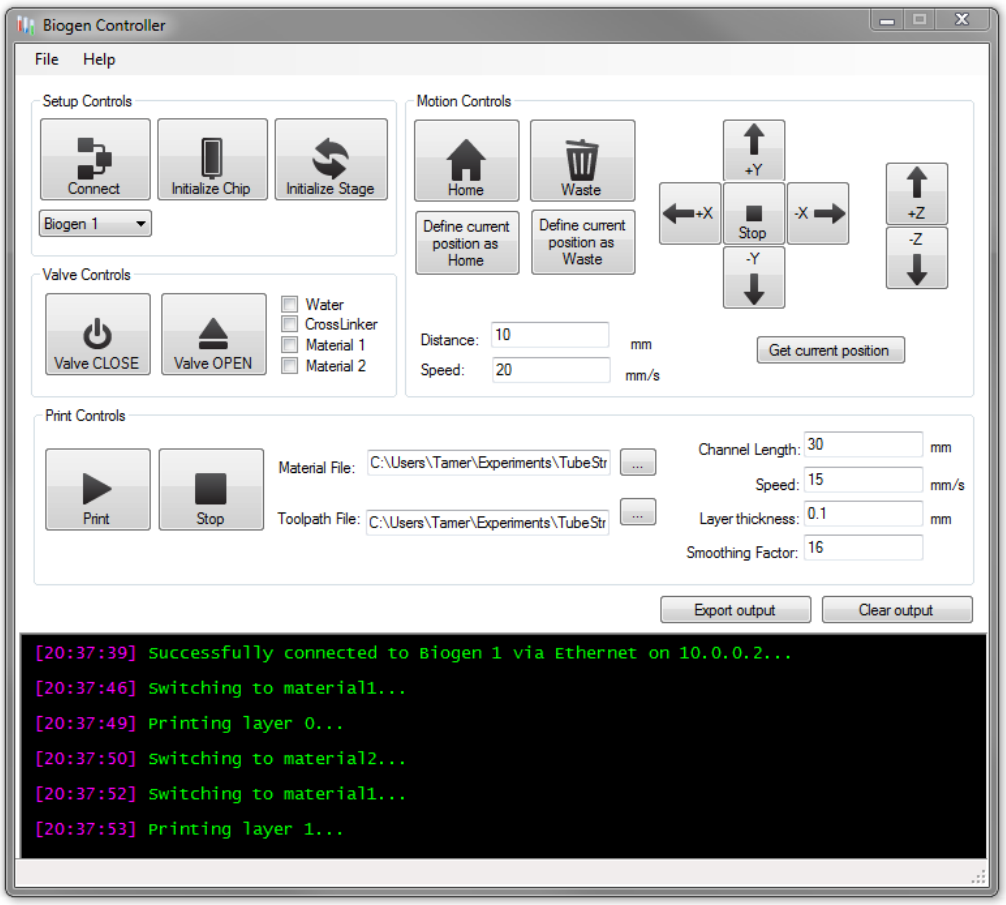


Figure 29: *Printer control software GUI*

Setup and initialization starts by first establishing a connection with the bioprinter. Then the positioning stage is initialized by moving all three axes to the limits and then returning back to the home position. The “Initialize Chip” button calls the initialization procedure outline in Table 6.

Motion control features allow direct control of the 3D bioprinter. Each axis could be moved independently by specifying the distance and the speed. Home and waste bin positions can be defined and saved. This allows users to set these two positions at the beginning of a printing experiment and return to these special positions at any time. The home position is typically the starting position of the print job and the waste bin position is usually the position where the printhead produces waste material during initialization. To set the initial home position of the nozzle prior to printing, the printhead was moved down until contact was made with the printing substrate and the nozzle was in the center of the printing substrate. In addition to motion controls, each valve on the printhead could be turned on or off using the GUI. Motion and valve controls are useful in testing printhead functionality and for debugging purposes.

The print controls section of the printer control software handles print jobs. The software takes material and toolpath files generated by Tissue Designer as inputs in order to print 3D tissue structures. The multi-axis move commands specified in the toolpath file are sent to the microcontroller’s on-board buffer which can store up to 512 incremental move segments. It is important to guarantee continuous motion during a print job in order to prevent material building up unevenly in one particular location of the structure. To ensure continuous motion and to allow for the printing for structures

that require more than 512 movement segments, the microcontroller's on-board buffer must never be emptied until the toolpath's last movement is executed. Otherwise, motion will be halted immediately after the execution of the last incremental move segment, even if the entire toolpath has yet to be fully executed. At the same time, it is important to not send commands if the on-board buffer is full to avoid buffer overflow. To solve this producer-consumer communication problem (shown in Figure 30), a dedicated thread is given the task of monitoring the status of the on-board buffer. As movements are executed and space becomes available in the buffer, commands are sent to the controller.

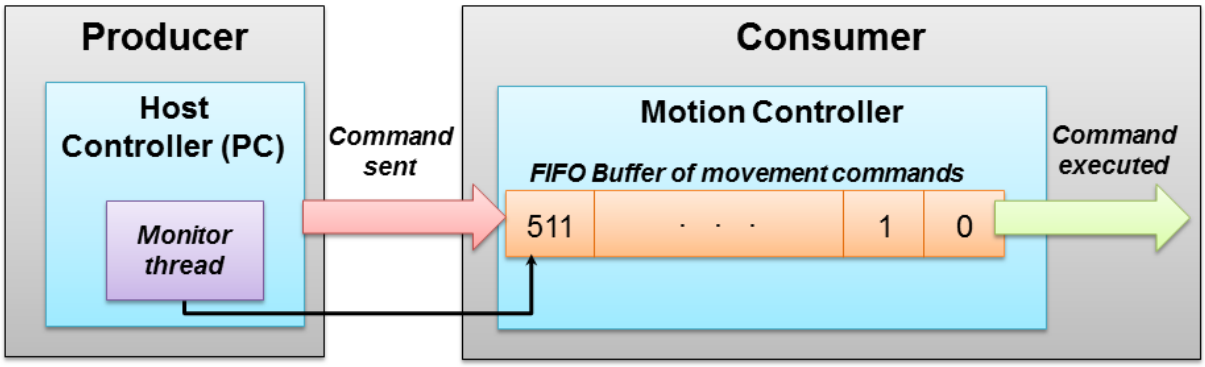


Figure 30: *Sending and executing move commands using buffer and monitor thread*

To handle material switching, the printer control software first reads the input material file and stores the material switch points as well as the corresponding material type in a queue of material objects. A dedicated thread is used to monitor an on-board register that stores the total distance traveled from the start of a print job. If the material

switch point of the material object stored at the front of the queue minus the channel length equals the distance traveled, then output signals are sent from the microcontroller to the pneumatic solenoid switch bank in order to switch to the specified material. As materials are switched, material objects are pulled from the front of the material queue. To parameterize this process and allow our software to be compatible with different printhead architectures, the channel length is one of the inputs of the printer control software. This flowchart for this synchronization process is shown in Figure 31.

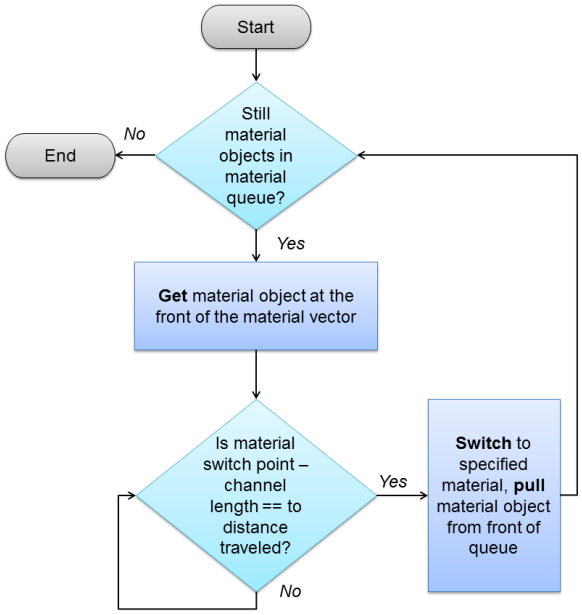


Figure 31: Flowchart for synchronization between printhead and toolpath

With the addition of new hardware modules (as shown in Figure 32), other parameters could be introduced to account for task-specific delays. For example, some

types of materials such as collagen may need to be printed under controlled-temperature conditions to prevent premature gelation and printhead clogging. New control parameters could be introduced to account for the time it takes to reach the temperature set point and this time could be synchronized to motion events in a similar approach to that used in handling *a priori* material switching. Moreover, new tags that define parameters such as temperature and concentration could be added to PTDL to account for these additional building block properties and printing specifications. In this way, the printhead could be thought of as a programmable lab-on-a-printer.

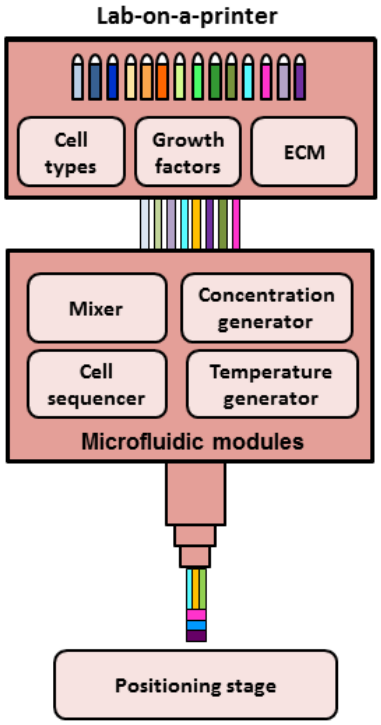


Figure 32: *Lab-on-a-printer schematic*

2.3 Summary

Prior to the development of our 3D bioprinting software toolchain, we were limited to printing simple structures. To print a desired structure, a user would have to manually program, line-by-line, the necessary printing instructions. This process can be extremely time consuming, tedious, and error-prone. Moreover, knowledge about the specific motion control programming and toolpath generation theory is required to produce fine process planning. One of the primary objectives of this work was to automate the process of printing 3D heterogeneous tissue structures. This would allow users, such as biologists and tissue engineers, to quickly design, visualize, graphically simulate, and then 3D print customizable tissue structures.

This chapter presented an overview of our novel 3D bioprinting system capable of creating customizable 3D heterogeneous tissue constructs based on user defined 3D models. An overview of the hardware components as well as detailed explanations of the design and implementation of our 3D bioprinting software toolchain was described. Our 3D bioprinting software toolchain is built around a novel Tissue Building Block Design (TBBD) method. Using the TBBD method, which is the core of our 3D bioprinting software toolchain, each tissue building block could be independently modeled using CAD software and then sliced according to a unique set of CAM specifications and assigned unique properties. The developed 3D bioprinting software toolchain easily integrates CAD and CAM, whereas the manual programming approach does not intuitively facilitate this. Printing complex heterogeneous structures is no longer

bottlenecked by the ability to design these structures in software, but mostly by hardware and material limitations. Due to these significant improvements in the functionality, efficiency, integration, automation, and control of the 3D bioprinting process, tissue development could be greatly accelerated. To encourage the collaborative development and sharing of tissue design files amongst users, the developed Tissue Designer software also supports a Printed Tissue Description Language (PTDL) used to describe the designed tissue structure. An online platform called OpenVitro was initiated to facilitate the sharing of tissue design files, including PTDL files. Details of the control over on-chip behavior and synchronization between printhead function and stage motion as well as printer communication was also explained in this chapter.

Chapter 3: Experiments, Results, and Discussion

In this chapter, we present and discuss our experiments and results. In the first section, we evaluate the performance of our implementation of the TBBD method by analyzing execution times. In the second section, we analyze and discuss the effect of design choices and printing parameters on the overall printing process and the challenges associated with our microfluidics-based method of bioprinting. In the third section, we demonstrate the functionality and asses the capabilities of the 3D bioprinting software toolchain by printing several different heterogeneous hydrogel structures using our 3D bioprinter.

3.1 Software Evaluation and Performance

To verify the functionality and evaluate the performance of our implementation of the TBBD method, we ran several simulations independent of our bioprinting system. The functionality was assessed by using a test structure and visualizing the generated slices. The performance was assessed by evaluating the speedup achieved using our TBBD method. The results were collected using an Intel Core i5-3210M processor running at 2.5 GHz and 8 GB of memory.

For the first set of simulations, our goal was to verify the functionality of our implementation of the TBBD method before printing. The verification was done by merging a set of sliced building blocks and plotting the generated point clouds of each

slice. For example, one of the test structures used in the verification process is shown in Figure 33. The structure is composed of nine building blocks: eight small cubes embedded within a larger cube. Each building block was designed in CAD software and independently sliced using different CAM specifications. Using Tissue Designer, each sliced building block was then assigned a different material number. Then the set of sliced tissue building blocks were prioritized and the entire tissue structure was compiled for printing using our core TBBD algorithm. The toolpaths of the first layer within the four different regions are shown in Figures 34-37. As shown, the results verify that the implementation of the TBBD method works correctly.

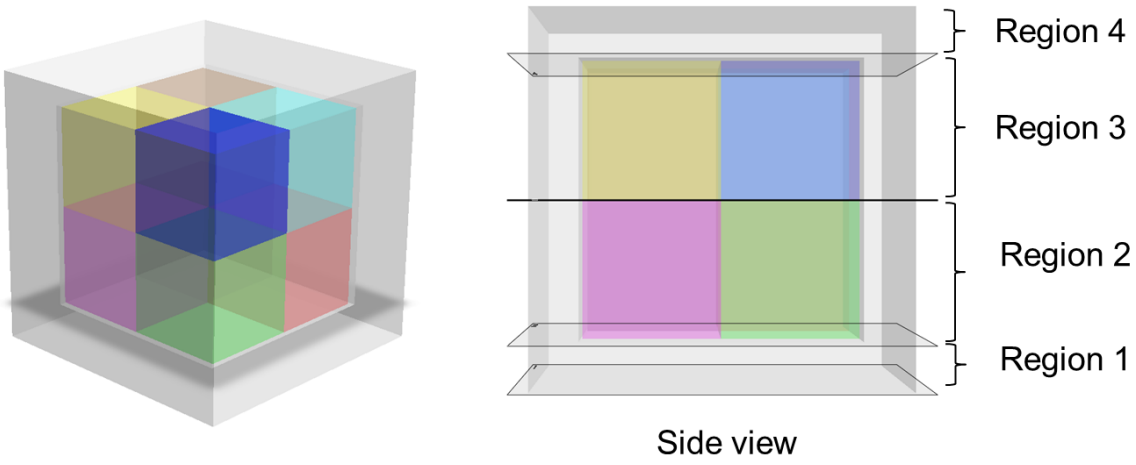


Figure 33: *Test structure composed of nine building blocks*

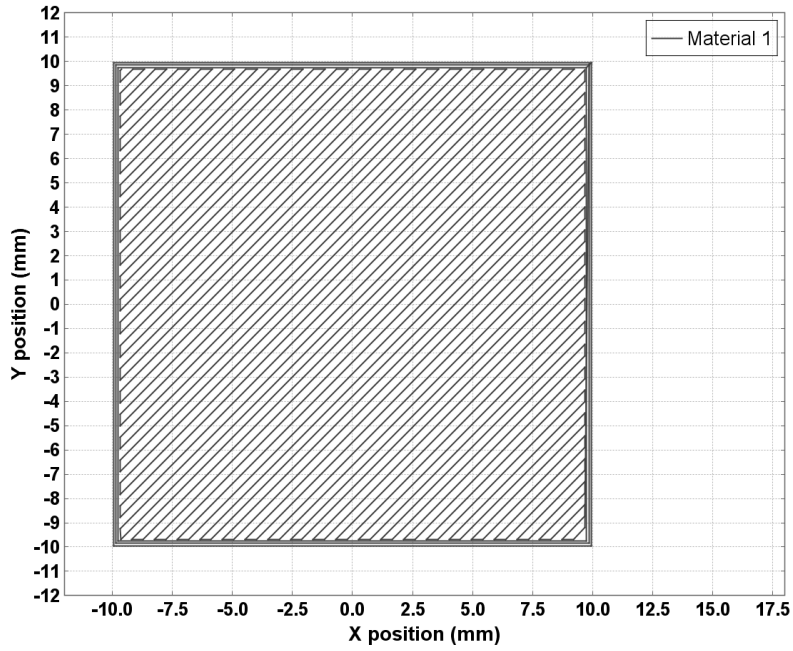


Figure 34: *Toolpath of first slice in region 1 of test structure*

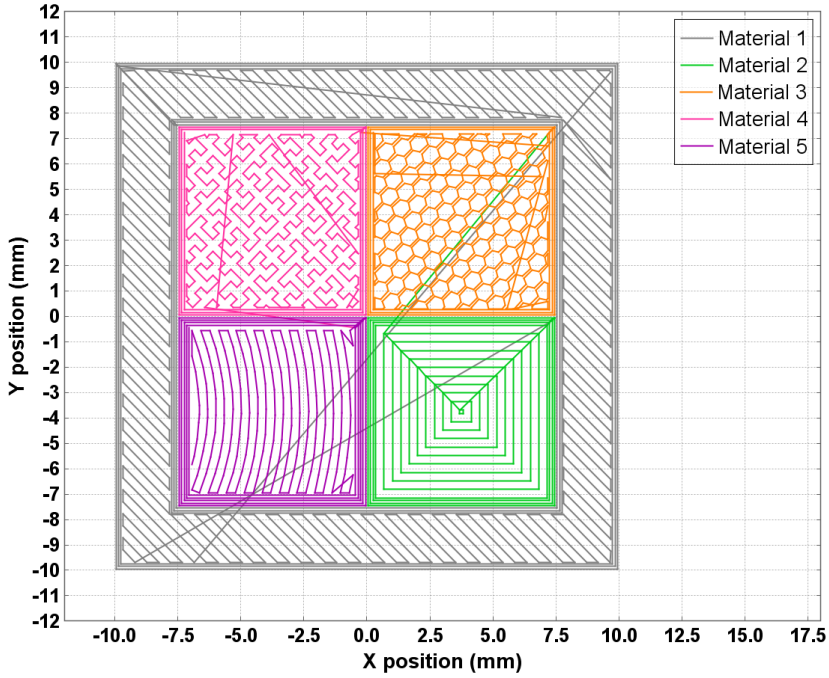


Figure 35: *Toolpath of first slice in region 2 of test structure*

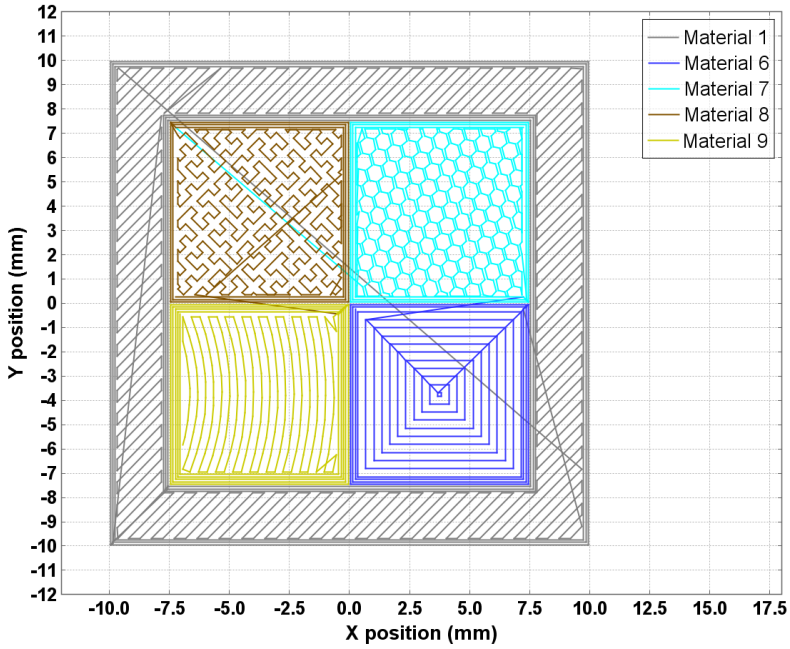


Figure 36: *Toolpath of first slice in region 3 of test structure*

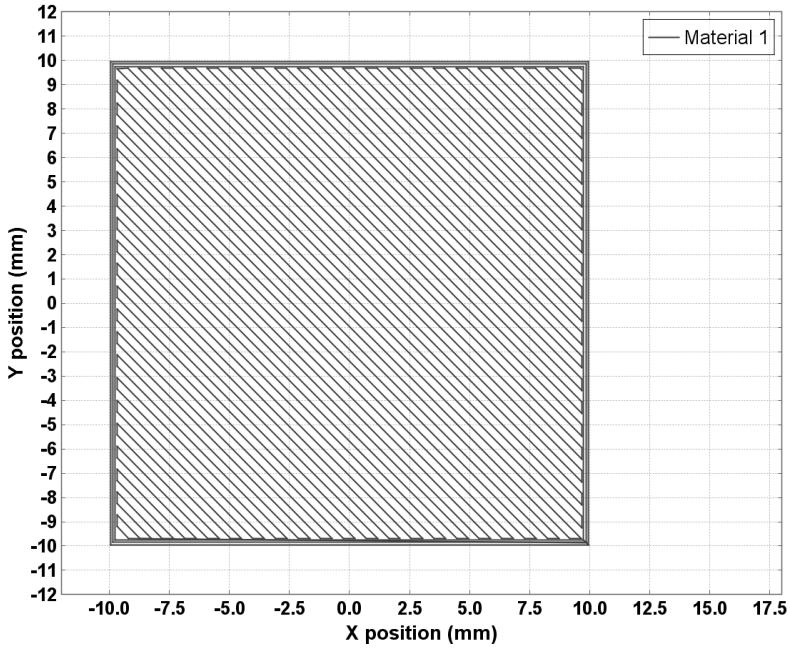


Figure 37: *Toolpath of first slice in region 4 of test structure*

Using a more standard design approach, a tissue model is sequentially sliced entirely, without avoiding redundant slicing computations, and then merged. Whereas using the TBBD method, a model can be granulated into identical surface geometries and only one of the granular building blocks is sliced to avoid redundant slicing. In other words, a tissue structure is decomposed into a set of simpler building blocks and then merged using our software. For the second set of simulations, our goal was to benchmark the performance of our developed TBBD algorithm versus a more standard design approach. To do this, the slicing and merging execution times were analyzed for various assembled models. As a demonstration of this, a cylinder with a diameter and height of 20 mm was chosen as a test structure. Using a more standard design method,

the tissue structure was merged using a single cylinder building block (with a height of 20 mm), relying heavily on the slicer software. Using the TBBD method, the tissue structure was merged using 100 identical cylinder building blocks, each with a height of 0.2 mm. All the building blocks were sliced using common CAM specifications: 100 µm layer thickness and extrusion width, 3 perimeters, 30% fill density, and varying fill patterns. The slicing, merging, and total execution times using both design approaches for different fill patterns are shown in Figures 38-40. As can be seen from the figures, the total execution time is reduced by avoiding redundant slicing while merging time stays about the same using both design approaches.

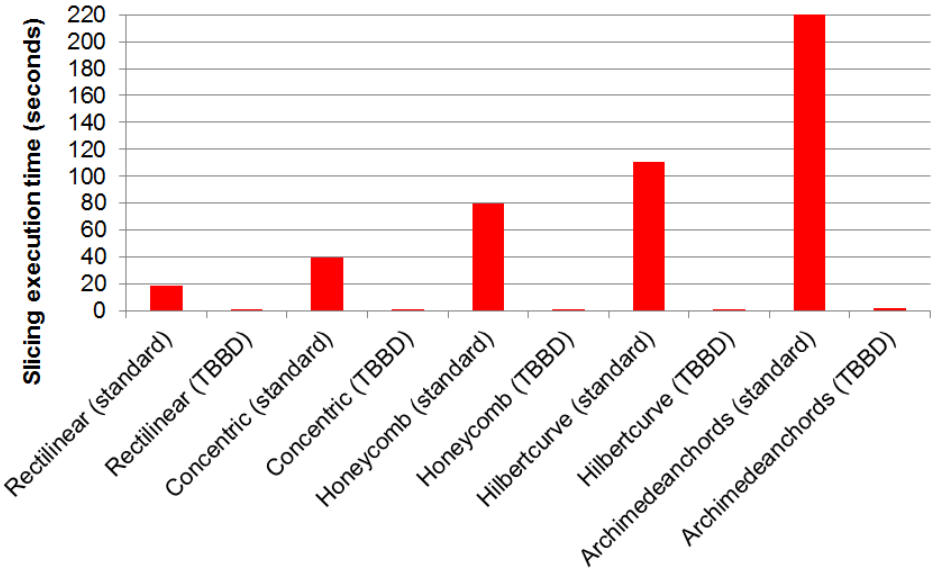


Figure 38: *Slicing execution times for both the standard design approach and the TBBD method for different fill patterns*

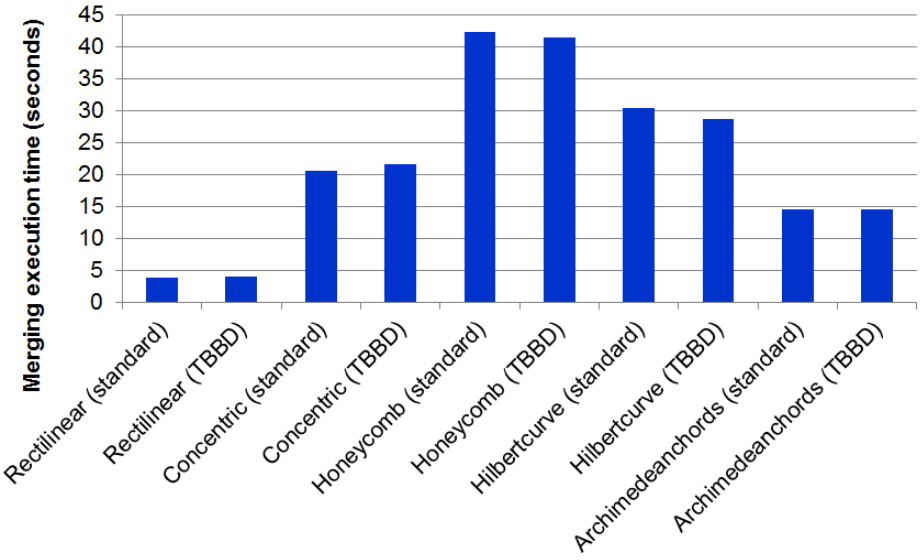


Figure 39: *Merging execution times for both the standard design approach and the TBBD method for different fill patterns*

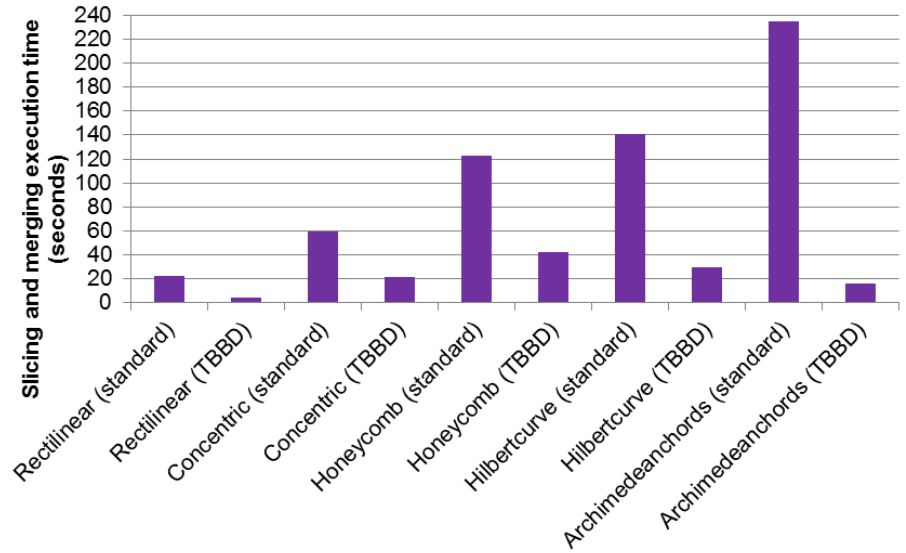


Figure 40: *Total execution times for both the standard design approach and the TBBD method for different fill patterns*

TBBD speedup over the more standard design method for various heights and fill patterns of the test structure is summarized in Figure 41. The results show that the TBBD method is as fast if not faster than the more standard design approach. Using the TBBD method, even though we are optimizing the slicing process through sliced building block reuse, our speedup is always limited by the merging execution time. Because of this, when we analyze the speedup curve, it is clear that the speedup saturates as the merging time becomes the bottleneck. However, the disadvantage of the TBBD method is that it relies on the user to granulate the desired structure into simpler building blocks that are then used to assemble the desired structure. This may be a demanding process if there is extreme variation in the slices in the Z build direction. Look-ahead algorithms could be developed in the future to reduce redundant slicing. Furthermore, the absolute time savings may not be significant for small structures, but for larger structures that are sliced with computationally intensive space filling curve algorithms (such as the archimedeanchords), the absolute time savings may be substantial. Moreover, adding more building blocks results in larger PTDL file sizes. However, since PTDL is XML-based, XML compression tools could be used if storage space becomes a major concern.

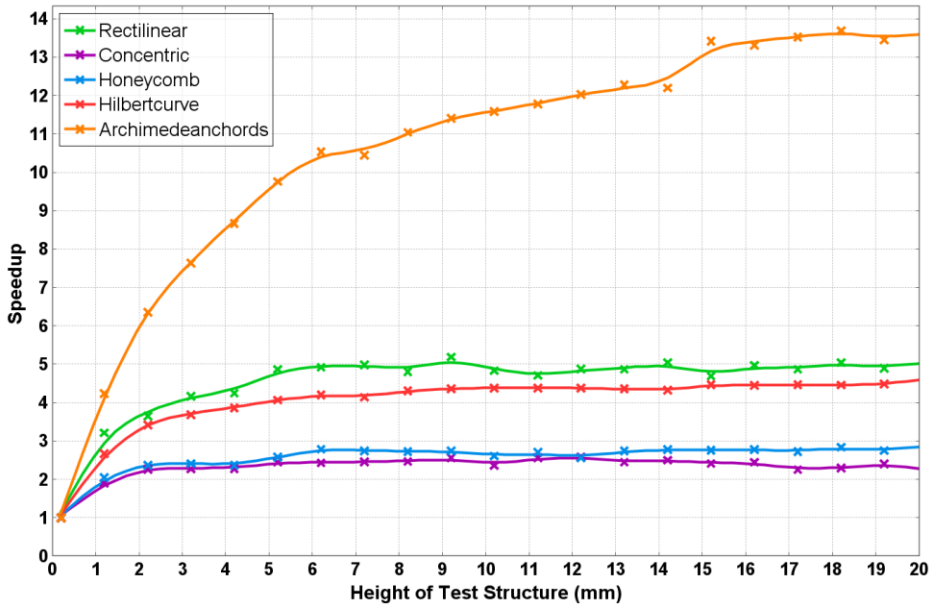


Figure 41: *Speedup using TBBD*

3.2 Design Choices, Printing Parameters, and Printing Process

Several experiments were conducted to analyze the effect of design choices, printing parameters, and the overall printing process. All the printing experiments were conducted using 1 wt% alginate (Protanal LF 120 M) and gelled using a 125 mM calcium chloride solution, unless otherwise stated. Additionally, during our experiments, we encountered several challenges related to our specific microfluidics-based bioprinting approach. The results of the conducted experiments, as well as the different

methods used to investigate the encountered challenges, will be discussed in this section.

Prior to printing a structure, the printing speed and fiber flow rate were calibrated. Dispensing too much material (fiber flow rate > printing speed) led to printhead clogging. Dispensing too little material (fiber flow rate < printhead speed) caused the fiber to be tugged and eventually break. Based on the most-recent printhead dimensions, the channel length parameter in the printer control software was set to 30 mm. Observed fiber coiling within the channel indicates that the fiber flow rate is faster than it should be in order to match the speed of the XY positioning stage. This leads to an effective channel length that is larger than the actual channel length, causing material buildup and unsynchronized material switching. This problem is demonstrated in Figure 42(a). To approximately match the fiber flow rate to the speed of the XY positioning stage given the current setup, the speed was gradually increased such that fiber coiling within the printhead was minimized (see Figure 42(b)). This visual calibration test was conducted at the start of a printing experiment to obtain the appropriate printing speed for particular printhead pressure values. Although the printhead pressure values driving each of the material input channels could be tuned to match a given speed, it was found that frequently changing printhead pressures led to problems such as clogging within the printhead. Therefore, once a pressure value was seen to produce a stable fiber, the appropriate printing speed was then found using the visual calibration test.

(a) (b)

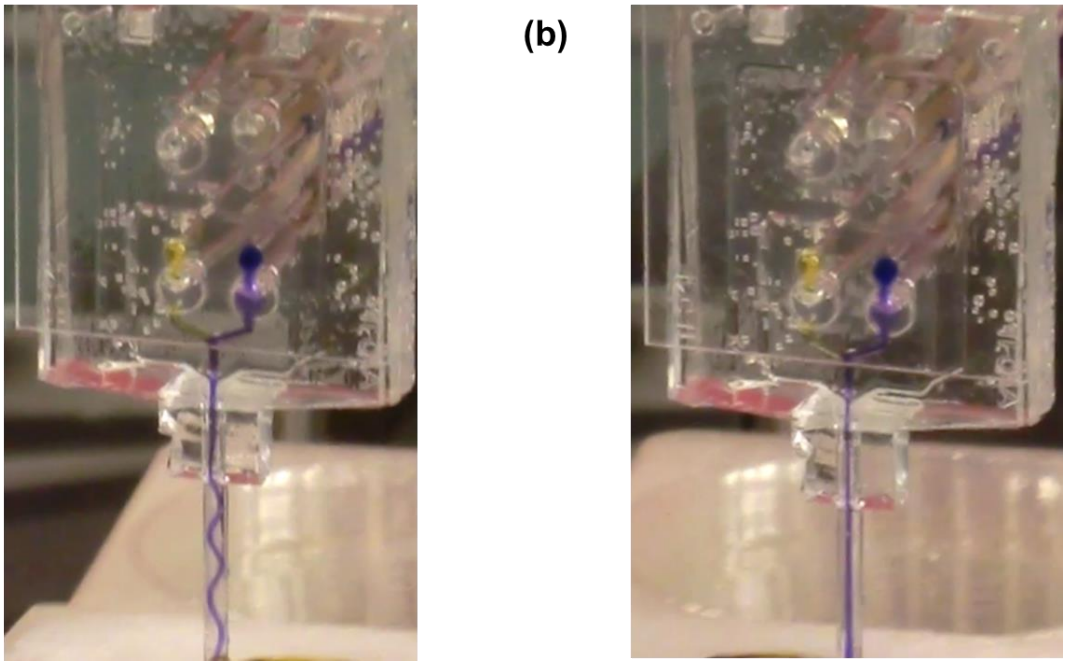


Figure 42: *(a) The printhead showing the fiber flow rate and speed are mismatched; fiber is coiled inside the printhead channel (b) The printhead showing the fiber flow rate and printing speed are matched; fiber is straight inside the printhead channel*

The effect of not accounting for the channel length was significant, particularly when printing structures with intra-layer material switching. When the channel length was not correctly set, material from one building block was incorrectly present in the adjacent building block. Whereas when the channel length was correctly set according to the printhead dimensions, significantly better results were obtained as shown in Figure 43.

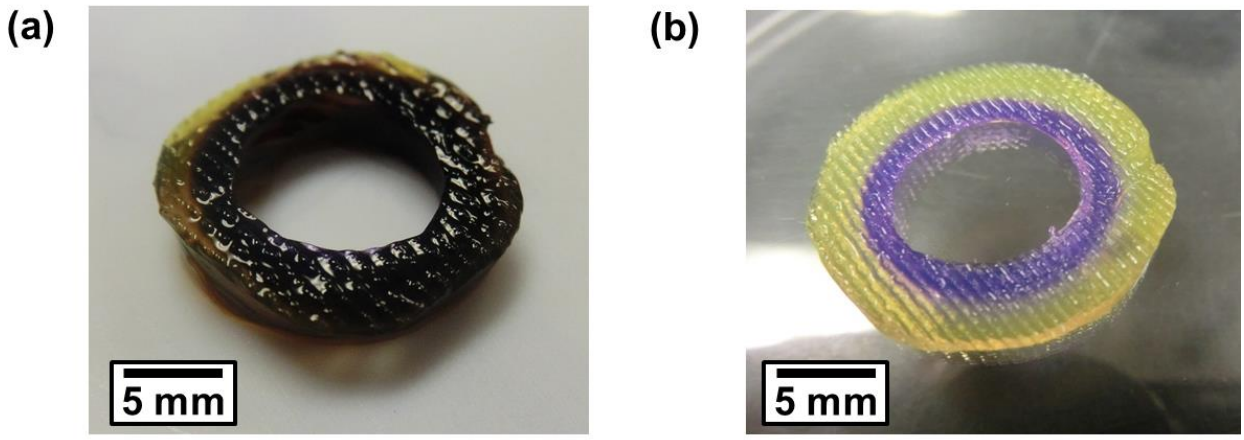
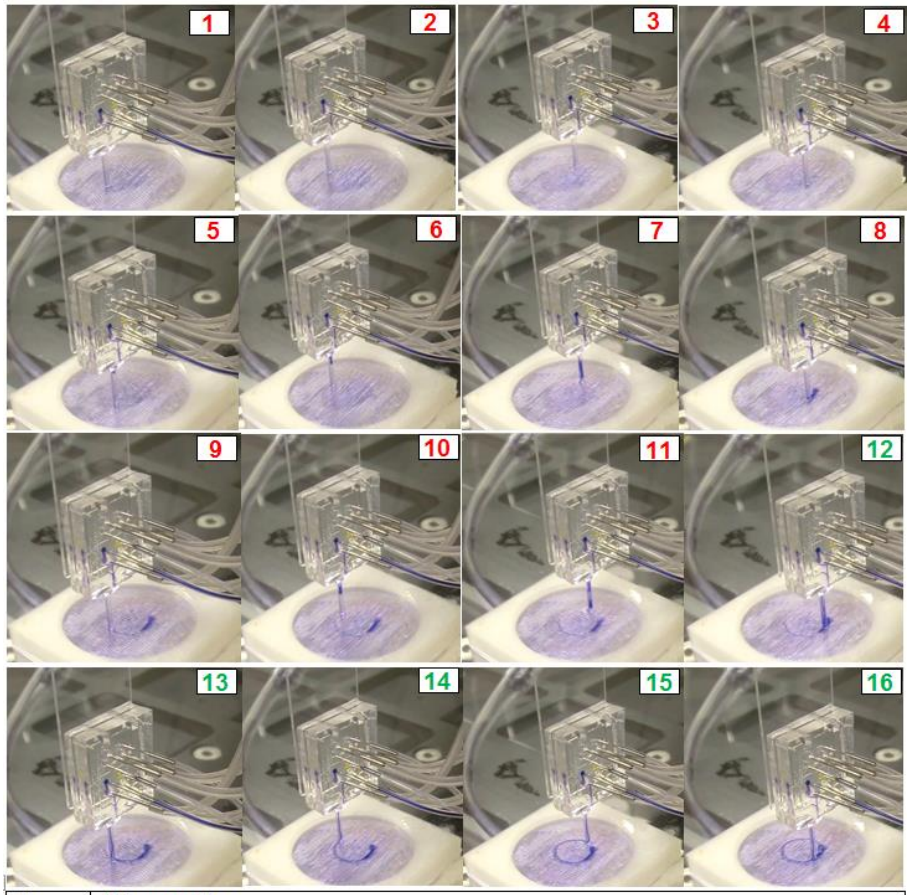


Figure 43: *(a) A multi-material coaxial tube structure without accounting for channel length during material switching (b) a multi-material coaxial tube structure with correctly set channel length*

As previously mentioned in Section 2.1, our printhead generates a solid hydrogel fiber by introducing a coaxially flowing crosslinking agent to a liquid pre-polymer. This gelation process is carried out within the channel length portion of the printhead shown in Figure 28. Because of this, there is a delay from the moment a request to start printing is invoked to the actual time that a fiber exits the printhead orifice. To observe and investigate this delay problem, a video was captured during the execution of a test printing session. The test printing session consisted of a clock-wise circular motion sequence. The video was captured until the moment a relatively uniform fiber was seen exiting the printhead orifice. Several frames from this video are shown in Figure 44, starting at a 9 o'clock position, then 12 o'clock, then 3 o'clock, then 6 o'clock, and so on. As can be seen, segments of the toolpath are executed without fiber deposition. This posed a challenge to the overall printing process. Through several printing experiments,

it was seen that a poorly printed first layer with gaps negatively affected overall print quality and structural integrity. Besides accounting for the delay associated with filling the channel length portion of the printhead with material, we also noticed that extra time was needed until a relatively uniform fiber exits the nozzle. To solve this problem, an initialization pattern was printed a distance away from the desired structure to help ensure that the printhead channel was loaded with material and that a relatively uniform fibre was exiting the printhead orifice prior to the first point of the desired toolpath. The initialization pattern could be designed as a separate building block and should be assigned the highest printing priority to ensure that it is scheduled to be printed before any subsequent building blocks. When designing the initialization structure, consideration should be taken to avoid interference with the desired structure's toolpath. For our printing experiments, the point clouds generated by Tissue Designer were graphed and analyzed prior to printing to avoid this problem.



#	Observation
1-5	One complete revolution is completed without fiber deposition
6	A fiber segment is seen at the top of the channel length
7	A discontinuous fiber segment is seen in the middle of channel length
8	A discontinuous fiber segment is deposited on the substrate
9	No fiber is seen in the channel length
10	A fiber segment is seen at the top of the channel length
11	Several discontinuous fiber segments are seen in the channel length
12	A continuous fiber is seen in the channel length
13-16	A continuous fiber is deposited on the substrate

Figure 44: *Several frames from a video of a test printing session in the absence of printing an initialization building block*

Besides its primary purpose, the use of an initialization structure served as a useful tool for determining if initial height of the nozzle was incorrectly set. If the nozzle was determined to be too close or too far from the substrate, the print job was halted

before completion. If the nozzle was too close to the printing substrate, the printhead clogged almost immediately due to material buildup in the nozzle and the nozzle tip dragging across the printing substrate. If the nozzle was too far from the printing substrate, a tapering effect in the build direction was observed because the layer-thickness would effectively be overestimated. An example of this effect is shown in Figure 45. In this example the input CAD file is a cube, but the printed structure appears pyramided. The tapering effect was also observed if the layer thickness was set too high.

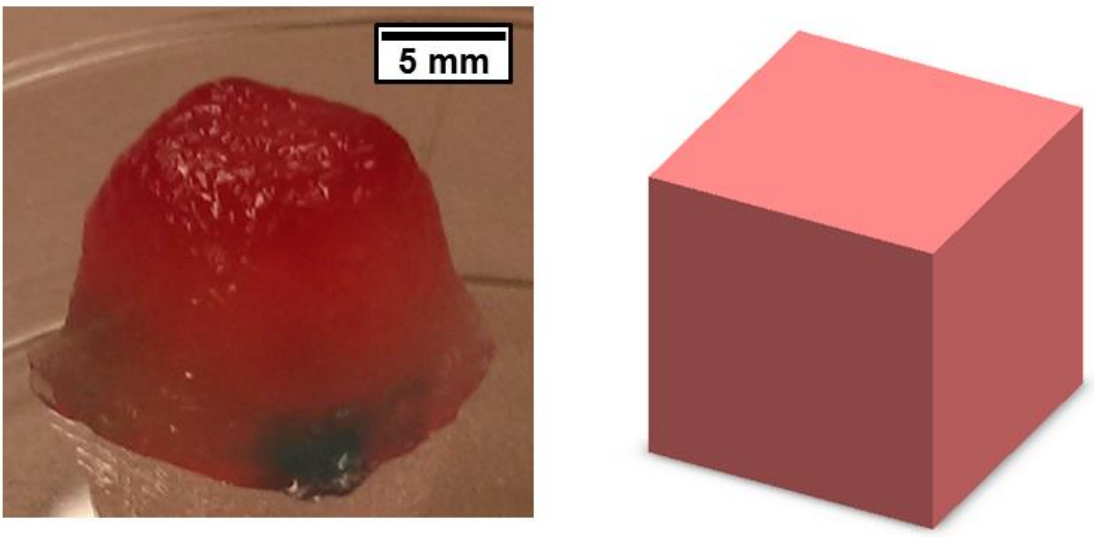


Figure 45: *Tapering effect due to incorrect nozzle leveling to printing substrate or incorrect layer thickness*

It was observed that when printing shapes with right edges such as squares, the corners were rounded as seen in Figure 46(a). To investigate this further, several printing experiments were conducted that involved printing 10x10 mm squares or variants under different conditions to see if corner rounding could be minimized. The first experiment involved modifying the square structure in CAD by dragging the corners outwards to offset the rounded corners. Using this technique, an improvement was observed, though at the cost of slightly warped sides, as can be seen in Figure 46(b). The second experiment involved pausing at corners for 500 milliseconds (Figure 46(c)). This corner slowdown technique failed due to material build up, clogging, and discontinuous fiber deposition caused by a mismatch between the calibrated printing speed and flow rate as previously set using the channel length visual calibration test mentioned earlier. To investigate this corner slowdown approach further, additional printhead and real-time pressure control functionality could be implemented to support changes in flow rate within a printing session. The third experiment involved printing a square pattern using a less viscous hydrogel. This was done by first diluting the alginate from 1 wt% down to 0.5 wt% and then printing a square structure. Using the less viscous alginate, corner rounding was reduced as seen in Figure 46(d), though at the cost of using a weaker fiber. The fiber was more malleable though the alginate appeared to be not fully gelled as can be seen by the blue dye leaching out of the fiber

onto the surface. This might also suggest that corner rounding could be due to the stiffness and the cylindrical nature of the fiber.

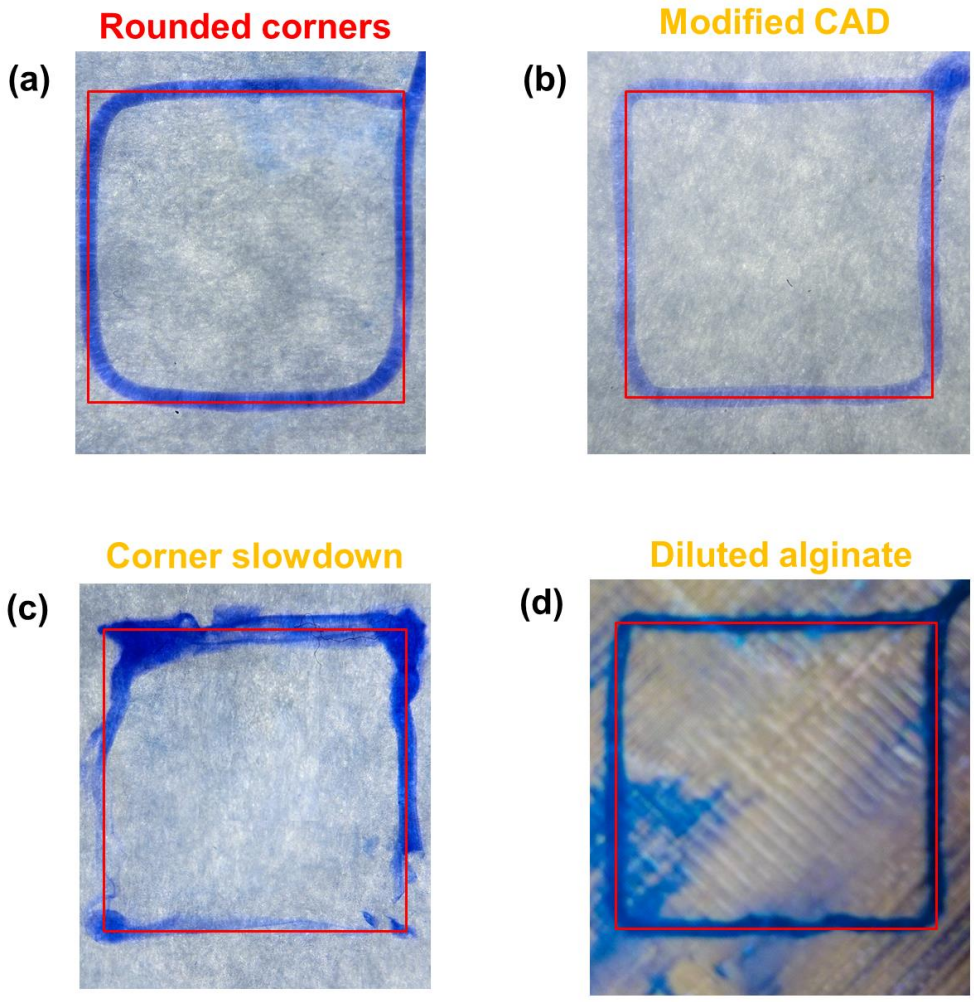


Figure 46: *(a) Printed square showing corner rounding effect (b) Printed CAD-modified square (c) Printed square using corner slowdown (d) Printed square using 0.5 wt % alginate*

Control over structural porosity may serve as a key feature to allow cell culture media to penetrate into the structure during the cell culturing step of the tissue engineering process as well to support removal of excess crosslinking agent during the printing step. The porosity of a structure can be optimized by tuning the fill density parameter (defined in Table 2). This parameter is specified by the user during the CAM phase of the tissue engineering process. It was seen that printing highly dense structures caused excessive pooling of crosslinking agent and eventual print failure. Several structures were printed using varying fill densities as well as fill patterns as shown in Figure 47. As can be seen in Figure 47, the fibers are not laid down exactly according to the generated patterns. The printed structures appear denser than the generated toolpaths. One reason behind this is a disparity between the extrusion width used in slicing the 3D CAD model and the actual fiber diameter. The extrusion width parameter should be accurately set to the actual fiber diameter. To address this issue, the printhead could be characterized further to generate a table of known alginate/crosslinker pressure ratios and the corresponding fiber diameter. The appropriate pressure values for each material channel could then be set by the pressure control system in order to produce a fiber with a diameter equal to the user-specified extrusion width. This control over fiber diameter could also help reduce the tapering effect previously mentioned as the actual layer thickness would be known *a priori*.

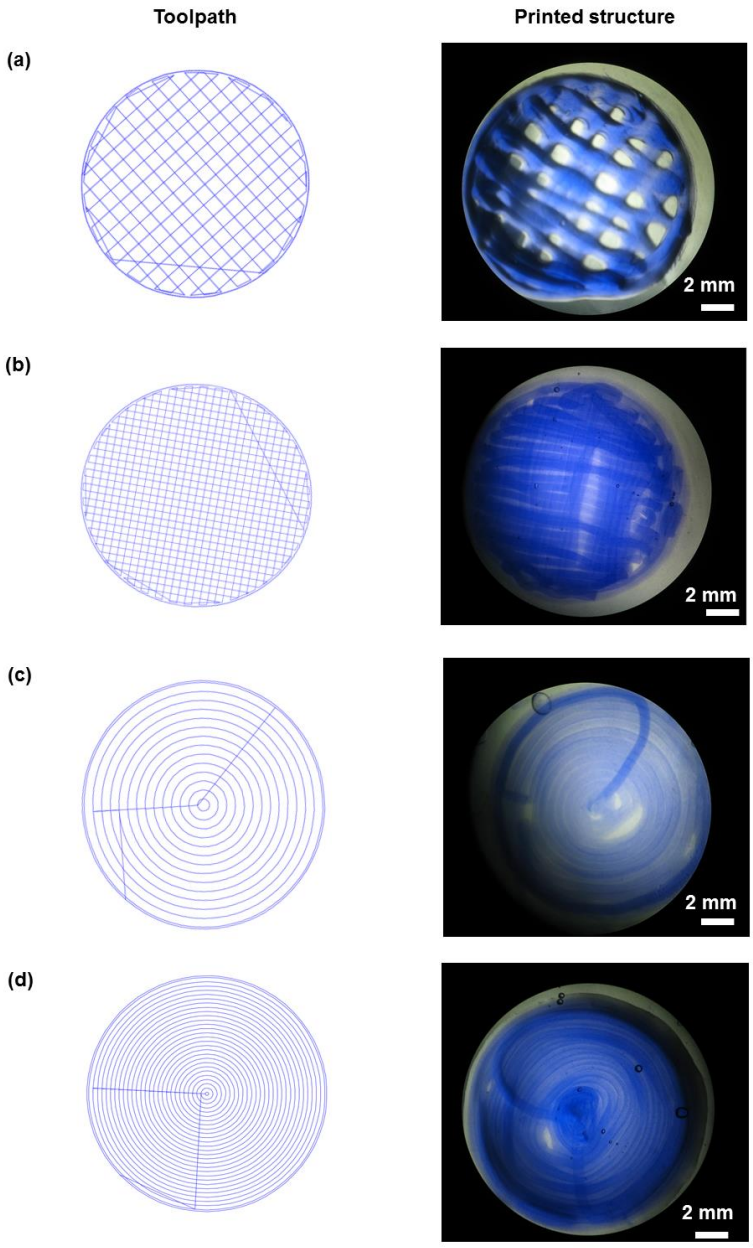


Figure 47**:** (a) Toolpath generated using 20% fill density and rectilinear patterning (b) toolpath generated using 40% fill density and rectilinear patterning (c) toolpath generated using 20% fill density and concentric patterning (d) toolpath generated using 40% fill density and concentric patterning

As an experimental demonstration of printing building blocks with varying infill patterns, the structure shown in Figure 48(a) was printed. The one-layer thick structure consists of two building blocks: a circular core and a circular shell. The circular core was sliced with 20% infill density and 90 degree rectilinear patterning; the circular shell was sliced with 20% infill density and concentric patterning. The pattern was printed using the microfluidic printhead (Figure 48(b)) as well as using a felt pen secured to the Z-axis (Figure 48(c)). In addition to a disparity between extrusion width and actual fiber width, the pooling of excess crosslinking agent as well as non-uniform fiber generation affected our ability to accurately recreate generated toolpaths. Pooling of the crosslinking agent caused many problems such as disturbing already deposited fiber, affecting fiber adhesion and stacking, and leading to material buildup and printhead clogging. A method to remove the excess crosslinking agent on-chip as opposed to relying completely on removal via a porous substrate after fiber deposition is something that can be investigated further to help address these problems.

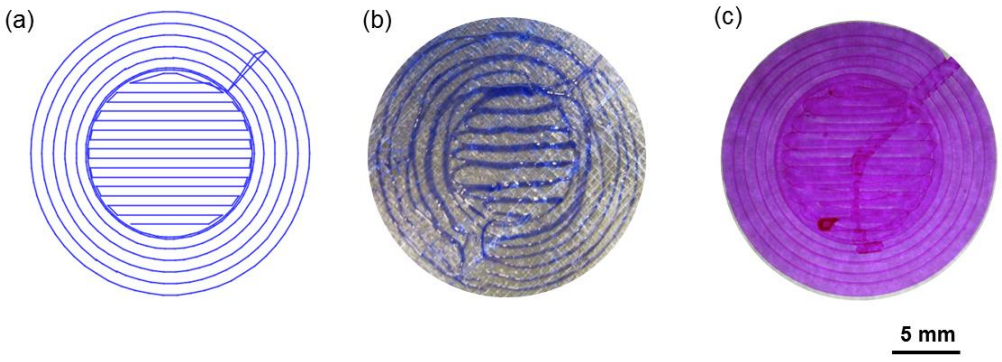


Figure 48: *Structure with varying infill patterning*

3.3 3D Printing Experiments

Several different structures were designed using Tissue Designer and then printed using the developed bioprinting system. The results show our ability to construct 3D hydrogel structures with arbitrary geometries and heterogeneous components. All the printed samples were composed of 1 wt% alginate and gelled using a 125 mM calcium chloride solution. The alginate was dyed using different colors to highlight multi-material building blocks. For these experiments, the printing speed, extrusion width, and layer thickness were often changed slightly because of variations in material composition, printhead fabrication, experimental setup, initial Z position of the nozzle, and channel pressures. For most of the printing experiments, the printing speeds ranged from 20-40 mm/s and the extrusion width and layer thickness ranged from 100-200 μm depending on the diameter of the generated fiber.

The first demonstrated printed structure was a heterogeneous tube. The building block used in the tissue design process was an STL file describing a tube segment. Since the multi-material tube structure is composed of a set of tube segments with identical surface geometries, only one tube segment building block was required. The building block was then sliced to generate the G-Code file. Using Tissue Designer, the same G-Code file was added multiple times to represent the stacked and identical tube segments, material information was assigned, and then the entire tissue structure was merged. The final printed structure is shown in Figure 49. These results highlight our ability to design and print structures with inter-layer material switching.

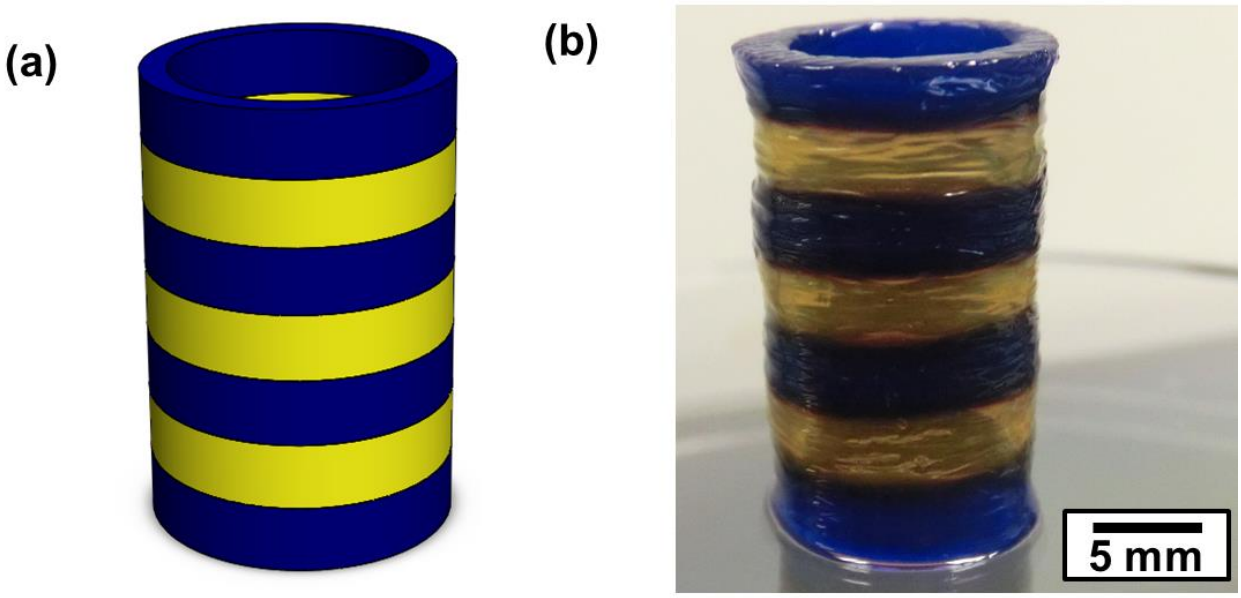


Figure 49: *(a) CAD representation of multi-material tube (b) printed multi-material tube*

By adding additional building blocks and using the scaling feature in Tissue Designer, a larger multi-material tube was designed and printed. This highlights our ability to quickly modify our tissue design files in order to print a different structure. The results are shown in Figure 50.

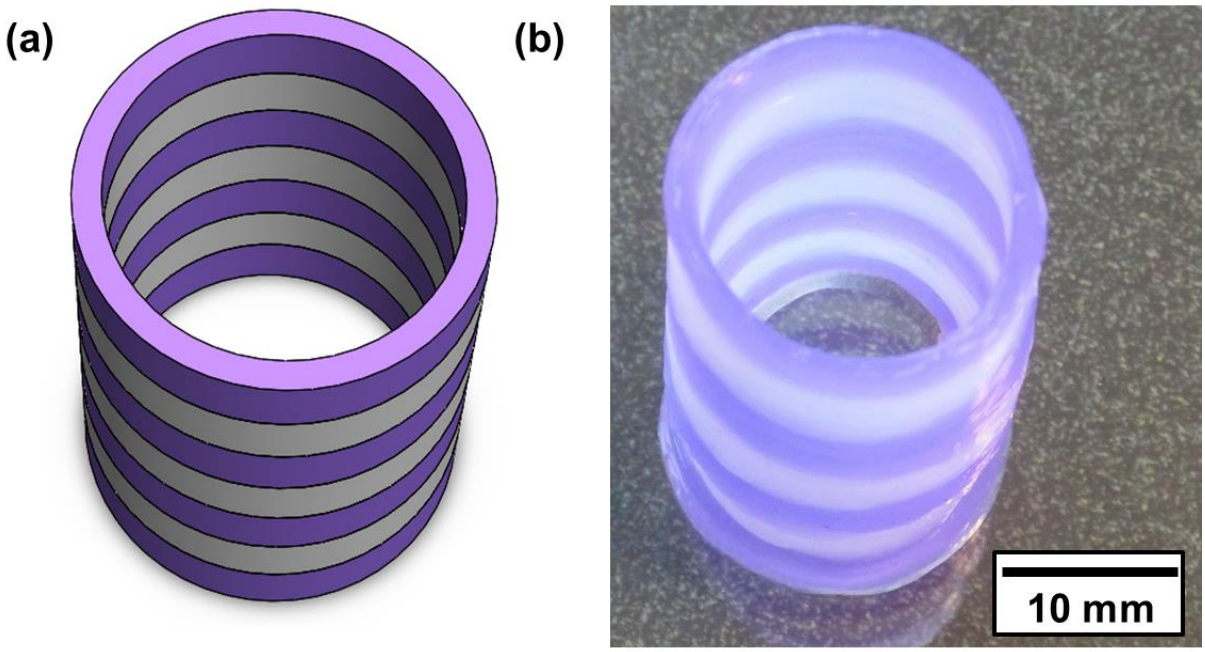


Figure 50: *(a) CAD representation of large multi-material tube (b) printed large multi-material tube*

The second printed structure was a heterogeneous coaxial tube. A blood vessel is a good example of such a structure. The coaxial tube structure was compiled using two tube building blocks. Each building block was assigned a different material number. The results are shown in Figure 51. These results demonstrate our ability to design and print structures with intra-layer material switching, allowing for the design of heterogeneous structures with lateral patterning.

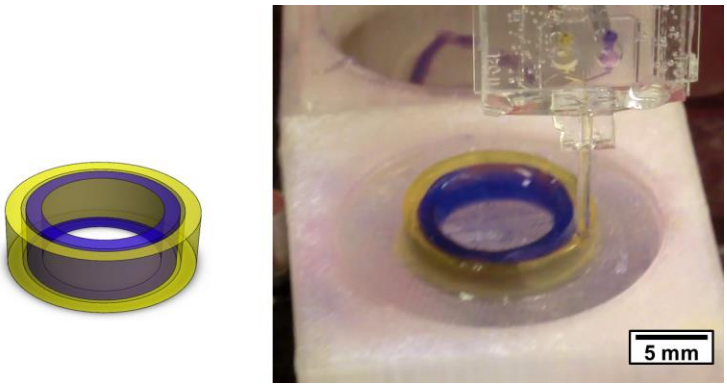


Figure 51: *(a) CAD representation of coaxial tube (b) Printed coaxial tube*

Using the scaling feature in Tissue Designer, a variant of the coaxial tube structure was designed. The inner tube was enlarged so that both tubes were brought closer to each other. This is shown in Figure 52.

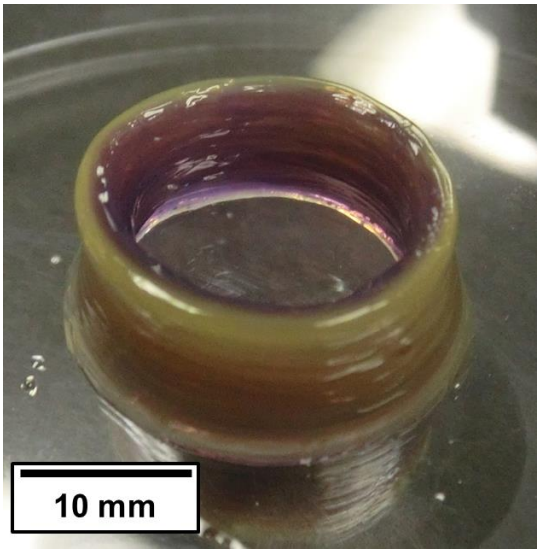


Figure 52: *Printed variant of coaxial tube*

As previously mentioned, it was observed that the first layer in the printed structure highly dictates the overall stability of the structure. Therefore, during our printing experiments we often printed an initialization pattern around our structure in order to ensure that the fiber was continuously stable before beginning to print our desired structure. As a result, in the case of the coaxial tube structure, we set the printing priorities such that slices from the outer tube are printed before slices from the inner tube. This was done to reduce trailing fibers between the concentric circles during travel movements, which is particularly important in the first layer.

The third printed structure was a multi-material cube composed of rectangular cube building blocks. As previously discussed in section 3.2, it was observed that sharp corners in a toolpath resulted in rounded corners in the final printed structure. To mitigate this problem in this case, the design of the corners was modified in CAD. The cube building block was sliced to produce layers with a single perimeter and filled with rectilinear patterning and a density of 40%. The sliced building blocks were stacked, assigned different material numbers, and merged to generate the necessary toolpath and material code. The results are shown in Figure 53. Since this structure was highly dense and several layers high, excess crosslinking agent pooled on the top most layer as the structure was built layer-by-layer and therefore the quality of fill patterning was affected.

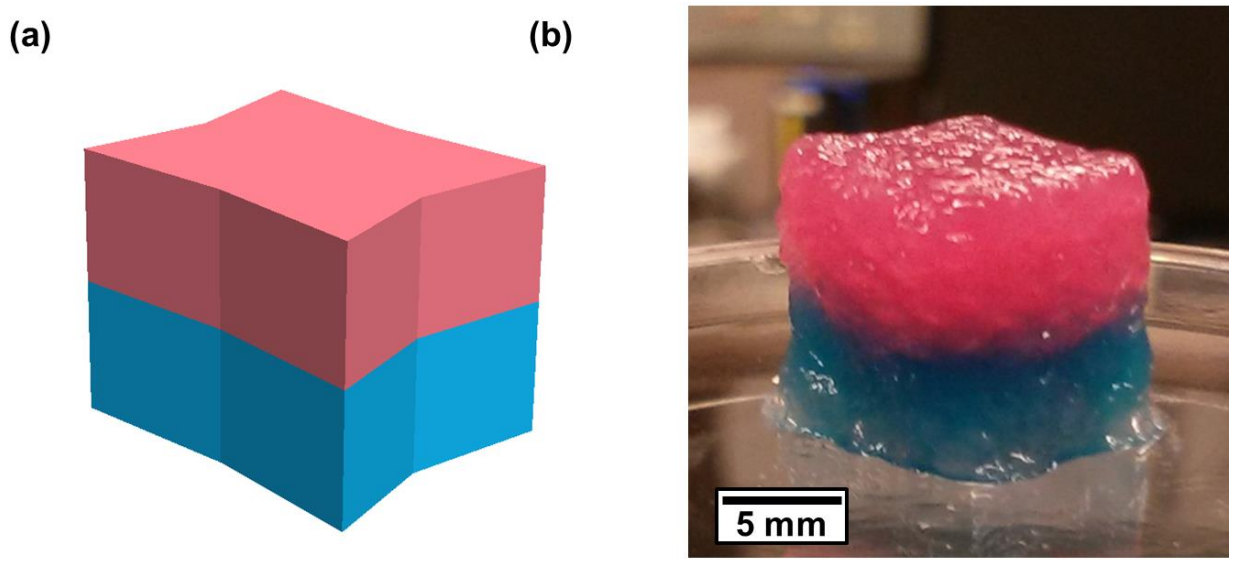


Figure 53: *(a) CAD representation of multi-material solid cube (b) printed multi-material solid cube*

Another hydrogel structure that was printed was the letters “UBC”. Because our current printhead design does not support being turned on and off during a print job, each letter was allowed to print to completion before starting the next letter. This was done to avoid trailing fibers between letters during travel movements. The printed structure is shown in Figure 54. Although the geometry of this structure isn’t really biological relevant, these results show our ability to design arbitrary 3D structures.

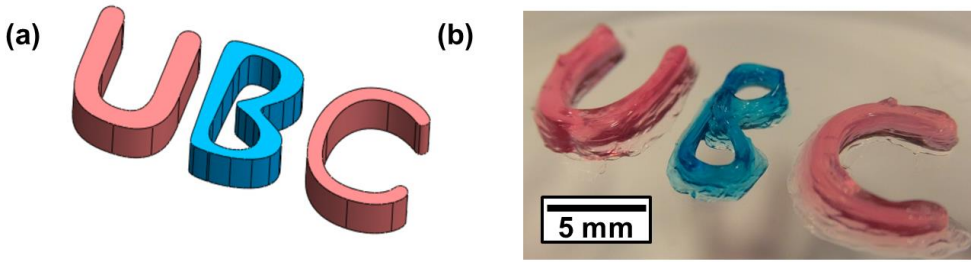


Figure 54**:** *(a) CAD representation of the letters "UBC" (b) printed "UBC"*

3.4 Summary

This chapter discussed the results from various experiments conducted to assess the performance of the developed software toolchain, investigate designs choices, printing parameters, and the overall printing process, demonstrate the 3D printing of different structures, and to validate basic 3D biology using our approach. Analyzing the TBBD speedup curves, we observed a combined slicing and merging execution time that is as fast as or faster than simply slicing and then merging the designed structure as one unit. Through several printing experiments, we were able to observe several problems such as unmatched flow rate and printing speed, unsynchronized material switching due to an incorrect channel length parameter, tapering effect due to incorrect layer thickness or nozzle/substrate leveling, poor first printed layer in the absence of an initialization pattern, corner rounding, and other challenges associated with the nature of the hydrogel and pooling of excess crosslinking agent. Methods and recommendations to mitigate or eliminate these problems were discussed. Using the TBBD method, we were able to successfully print several heterogeneous structures, including some with multiple materials.

Chapter 4: Conclusion and Future Work

Applying 3D printing technology to tissue engineering as a means of creating customized human tissue constructs is a relatively new idea that has the potential to revolutionize biological research. There are several research studies that demonstrate that 3D cell cultures better mimic real biological tissue than the standard 2D cell co- or mono-cultures. In the field of drug discovery, models that exhibit real tissue mechanisms under exposure to drugs are of significant interest. Using 3D tissue constructs to get a more realistic and representative response to a given drug could improve the efficiency and significantly reduce the cost of the drug discovery process, and ultimately replace animal models as a testing standard. For these reasons, we have developed a new platform capable of designing and implementing heterogeneous 3D biological structures based on user-defined 3D CAD models.

In this work, we developed a novel 3D bioprinting software toolchain capable of transforming user-defined 3D models into 3D printed biological structures with heterogeneous properties and a platform for a programmable lab-on-a-printer. This will potentially allow tissue engineers and biologists to rapidly design and 3D print advanced *in vitro* tissues that can be used in many applications such as drug discovery.

The 3D bioprinting software toolchain was built around a novel Tissue Building Block Design (TBBD) method that allows users to assemble structures with complex architectures and multiple properties from a set of simpler building blocks. Through the

TBBD method, each building block could be independently modeled using CAD software and then sliced according to a unique set of CAM specifications. A TBBD algorithm was developed to allow users to assign material types and other unique properties to each building block and then arrange the building blocks in 3D. The algorithm then generates the layered heterogeneous process planning required to print the designed multi-component tissue model. Interpreting and structuring G-Code in an object-oriented way makes our software flexible and scalable. The use of Flex and Bison compiler utilities allows for easier interoperability across different bioprinting systems and hardware implementations. Analyzing the TBBD speedup curves, we observed a combined slicing and merging execution time that is as fast as or faster than simply slicing and then merging the designed structure as one unit. An intuitive graphical user interface called Tissue Designer, built around the core merging algorithm, was developed. We established full automation and coordination between multi-functional printheads and stage motion. We also developed control systems and printing coordination techniques that are printer independent and could be adapted to other hardware implementations. To enable community development of printed tissues and to facilitate the exchange and reuse of the software-described tissues using an open standard, Printed Tissue Description Language (PTDL) was developed. An online platform called OpenVitro was initiated for the sharing of these tissue design files.

Using the developed 3D bioprinting software toolchain, we were able to successfully print several heterogeneous structures with both inter- and intra-layer material switching, validating our capability to reproduce the desired structure. Multiple

material printing is handled automatically; material switching is coordinated programmatically, in synchrony with toolpath motion. The approach used for *a priori* material switching could be used to account for delays associated with future printhead tasks to allow for properties to be applied in a controlled way along different points of a continuously printed fiber. We systematically optimized the printing process by considering print related issues such as unmatched flow rate and printing speed, printhead function and stage motion control and coordination, poor first layer printing, tapering effect and corner rounding.

Future work in this area could incorporate additional features in the PTDL format to accommodate for additional printhead functionality such as concentration and temperature generators. Additionally, information pertaining to material preparation, post-printing, and cell culturing stages of the tissue development process could also be added to the PTDL standard. Expanding PTDL would allow biologists and tissue engineers to exchange more detailed process information using a common language. This language expansion will require additional requirements that could be gathered from the OpenVitro community. To increase printing throughput, another possible direction would be to develop algorithms to control and synchronize a multi-nozzle printhead and print into a multi-well plate. This could be very useful in applications such as high-throughput drug screening where the responses of numerous drug compounds on cellular structures are tested. Using the TBBD method, future work could also investigate the effect of different internal architectures on biological response.

References

[1] I. Gibson, R. W. David, and B. Stucker, *Additive manufacturing technologies*. New York, NY: Springer, 2010.

[2] S. Emanuel, M. Cima, and J. Cornie, "Three-dimensional printing: rapid tooling and prototypes directly from a CAD model," *CIRP Annals-Manufacturing Technology*, vol. 39, no. 1, pp. 201-204, 1990.

[3] B. Evans, *Practical 3D Printers*. Berkely, CA: Apress, 2012.

[4] D. L. Bourell, J. B. Beaman, M .C. Leu, and D. W. Rosen, "A brief history of additive manufacturing and the 2009 roadmap for additive manufacturing: looking back and looking ahead," in *US-Turkey Workshop on Rapid Technologies*, Istanbul, Turkey, 2009, pp. 5-11.

[5] T. T. Wohlers, *Wohlers Report 2013: Additive Manufacturing and 3D Printing State of the Industry: Annual Worldwide Progress Report*. Colorado: Wohlers Associates, Inc. , 2013.

[6] J. P. Kruth, M. C. Leu, and T. Nakagawa, "Progress in additive manufacturing and rapid prototyping," *CIRP Annals-Manufacturing Technology*, vol. 47, no. 2, pp. 525-540, 1998.

[7] C. Mota, "The rise of personal fabrication," in *Proceedings of the 8th ACM conference on Creativity and cognition*, Atlanta, GA, 2011, pp. 279-288.

[8] J.M. Pearce, "Building research equipment with free, open-source hardware," *Science*, vol. 337, no. 6100, pp. 1303-1304, 2012.

[9] E. Malone and H. Lipson, "Fab@ Home: the personal desktop fabricator kit," *Rapid Prototyping Journal*, vol. 13, no. 4, pp. 245-255, 2007.

[10] G. Stemp-Morlock, "Personal fabrication," *Communications of the ACM*, vol. 53, no. 10, pp. 14-15, 2010.

[11] T. J. Horn and O. Harrysson, "Overview of current additive manufacturing technologies and selected applications," *Science progress*, vol. 95, no. 3, pp. 255-282, 2012.

[12] S. H. Huang, P. Liu, A. Mokasdar, and L. Hou, "Additive manufacturing and its societal impact: a literature review," *The International Journal of Advanced Manufacturing Technology*, vol. 67, no. 5-8, pp. 1191-1203, 2013.

[13] B. R. Ringeisen, B. J. Spargo, and P. K. Wu, *Cell and Organ Printing*. New York: Springer, 2010.

[14] J. A. DiMasi, L. Feldman, A. Seckler, and A. Wilson, "Trends in risks associated with new drug development: success rates for investigational drugs," *Clinical Pharmacology & Therapeutics*, vol. 87, no. 3, pp. 272-277, 2010.

[15] Pharmaceutical Research and Manufactures of America, *PhRMA 2012 Profile.* Washington, DC: PhRMA, 2012.

[16] A. Abbott, "Cell culture: biology's new dimension," *Nature*, vol. 424, no. 6951, pp. 870-872, 2003.

[17] M. W. Tibbitt and K. S. Anseth, "Hydrogels as extracellular matrix mimics for 3D cell culture," *Biotechnology and bioengineering*, vol. 103, no. 4, pp. 655-663, 2009.

[18] N.T. Elliott and F. Yuan, "A review of three-dimensional in vitro tissue models for drug discovery and transport studies," *Journal of pharmaceutical sciences*, vol. 100, no. 1, pp. 59-74, 2011.

[19] K. M. Yamada and E. Cukierman, "Modeling tissue morphogenesis and cancer in 3D," *Cell*, vol. 130, no. 4, pp. 601-610, 2007.

[20] D. Huh, G. A. Hamilton, and D. E. Ingber, "From 3D cell culture to organs-on-chips," *Trends in cell biology*, vol. 21, no. 12, pp. 745-754, 2001.

[21] B. Starly and W. Sun, "Internal scaffold architecture designs using lindenmayer systems," *Computer-Aided Design and Applications*, vol. 4, no. 1-4, pp. 395-403, 2007.

[22] S. M. Giannitelli, D. Accoto, M. Trombetta, and A. Rainer, "Current trends in the design of scaffolds for computer-aided tissue engineering," *Acta biomaterialia*, vol. 10, no. 2, pp. 580-594, 2014.

[23] I. Zein, D. W. Hutmacher, K. C. Tan, and S.H. Teoh, "Fused deposition modeling of novel scaffold architectures for tissue engineering applications," *Biomaterials*, vol. 23, no. 4, pp. 1169-1185, 2002.

[24] Y. Wang and J.P. Duarte, "Automatic generation and fabrication of designs," *Automation in construction*, vol. 11, no. 3, pp. 291-302, 2002.

[25] P. Kulkarni, A. Marsan, and D. Dutta, "A review of process planning techniques in layered manufacturing," *Rapid Prototyping Journal*, vol. 6, no. 1, pp. 18-35, 2000.

[26] R. Jones et al., "RepRap--the replicating rapid prototyper," *Robotica*, vol. 29, no. 1, pp. 177-191, 2011.

[27] B.T. Wittbrodt et al., "Life-cycle economic analysis of distributed manufacturing with open-source 3-D printers," *Mechatronics*, vol. 23, no. 6, pp. 713-726, 2013.

[28] G. Boothroyd, "Product design for manufacture and assembly," *Computer-Aided Design*, vol. 26, no. 7, pp. 505-520, 1994.

[29] V. Kumar and D. Dutta, "An assessment of data formats for layered manufacturing," *Advances in Engineering Software*, vol. 28, no. 3, pp. 151-164, 1997.

[30] B. Valentan, T. Brajlih, I. Drstvensek, and J. Balic, "Basic solutions on shape complexity evaluation of STL data," *Journal of Achievements in Materials and Manufacturing Engineering*, vol. 26, no. 1, pp. 73-80, 2008.

[31] P. J. Bártolo, *Stereolithography: materials, processes and applications*. New York: Springer, 2011.

[32] Y. Koren, *Computer control of manufacturing systems*. New York: McGraw-Hill, 1983.

[33] X. W. Xu and Q. He, "Striving for a total integration of CAD, CAPP, CAM and CNC," *Robotics and Computer-Integrated Manufacturing*, vol. 20, no. 2, pp. 101-109, 2004.

[34] T. Qiangping, W. Yu, and Q. Guorong, "A Middleware for Open CNC Architecture," in *IEEE Proceedings: International Conference on Automation Science and Engineering (CASE)*, Shanghai, 2006, pp. 558-561.

[35] T. Schroeder and M. Hoffmann, "Flexible automatic converting of NC programs. A cross-compiler for structured text," *International journal of production research*, vol. 44, no. 13, pp. 2671-2679, 2006.

[36] P. Smid, *CNC programming handbook*. New York: Industrial Press, 2008.

[37] J. W. Nichol and A. Khademhosseini, "Modular tissue engineering: engineering biological tissues from the bottom up," *Soft Matter*, vol. 5, no. 7, pp. 1312-1319, 2009.

[38] A. Tamayol et al., "Fiber-based tissue engineering: Progress, challenges, and opportunities," *Biotechnology advances*, vol. 31, no. 5, pp. 669-687, 2013.

[39] W. C. Wilson and T. Boland, "Cell and organ printing 1: protein and cell printers," *The Anatomical Record Part A: Discoveries in Molecular, Cellular, and Evolutionary Biology*, vol. 272, no. 2, pp. 491-496, 2003.

[40] D. L. Cohen, E. Malone, H. Lipson, and L. J. Bonassar, "Direct freeform fabrication of seeded hydrogels in arbitrary geometries," *Tissue engineering*, vol. 12, no. 5, pp. 1325-1335, 2006.

[41] C. Khatiwala, R. Law, B. Shepherd, S. Dorfman, and M. Csete, "3D Cell bioprinting for regenerative medicine research and therapies," *Gene Therapy and Regulation*, vol. 7, no. 1, p. 1230004, 2012.

[42] I. T. Ozbolat and Y. Yu, "Bioprinting toward organ fabrication: challenges and future trends," *IEEE transactions on bio-medical engineering*, vol. 60, no. 3, pp. 691-699, 2013.

[43] E. Kang et al., "Digitally tunable physicochemical coding of material composition and topography in continuous microfibres," *Nature materials*, vol. 10, no. 11, pp. 877-883, 2011.

[44] K. HoáLee, "Novel PDMS cylindrical channels that generate coaxial flow, and application to fabrication of microfibers and particles," *Lab on a Chip*, vol. 10, no.14, pp. 1856-1861, 2010.

[45] S. Shin et al., ""“On the fly” continuous generation of alginate fibers using a microfluidic device," *Langmuir*, vol. 23, no. 17, pp. 9104-9108, 2007.

[46] S. Ghorbanian, "Microfluidic probe for direct write of soft cell scaffolds," M.Eng. thesis, Department of Biomedical Engineering, McGill University, Montreal, Quebec, 2011.

[47] M. Vaezi, S. Chianrabutra, B. Mellor, and S. Yang, "Multiple material additive manufacturing--Part 1: a review: This review paper covers a decade of research on multiple material additive manufacturing technologies which can produce complex geometry parts with different materials," *Virtual and Physical Prototyping*, vol. 8, no. 1, pp. 19-50, 2013.

[48] S. Khalil and W. Sun, "Bioprinting endothelial cells with alginate for 3D tissue constructs," *Journal of biomechanical engineering*, vol. 131, no. 11, p. 111002, 2009.

[49] P. S. Maher, R. P. Keatch, K. Donnelly, R. E. Mackay, and J. Z. Paxton, "Construction of 3D biological matrices using rapid prototyping technology," *Rapid Prototyping Journal*, vol. 15, no. 3, pp. 204-210, 2009.

[50] M. Lang, W. Wang, X. Chen, and T. Woodfield, "Integrated system for 3D assembly of bio-scaffolds and cells," in *IEEE Proceedings: International Conference on Automation Science and Engineering (CASE)*, Toronto, Ontairo , 2010, pp. 786-791.

[51] J. P. Temple et al., "Engineering anatomically shaped vascularized bone grafts with hASCs and 3D-printed PCL scaffolds," *Journal of Biomedical Materials Research Part A*, 2014.

[52] K. Vidimče, S. Wang, J. Ragan-Kelley, and W. Matusik, "Openfab: A programmable pipeline for multi-material fabrication," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 136, 2013.

[53] C. Hsieh and N. A. Langrana, "A system approach in extrusion-based multi-material CAD," in *Solid Freeform Fabrication Symposium*, Austin, Texas, 2001, pp. 313-321.

[54] S. Khalil, J. Nam, and W. Sun, "Multi-nozzle deposition for construction of 3D biopolymer tissue scaffolds," *Rapid Prototyping Journal*, vol. 11, no. 1, pp. 9-17, 2005.

[55] J. Hiller and H. Lipson, "STL 2.0: a proposal for a universal multi-material Additive Manufacturing File format," in *Proc. Solid Freeform Fabrication Symposium (SFF'09), Austin,* Texas, 2009, pp. 266-278.

[56] W. D. Li, G. Q. Jin, L. Gao, C. Page, and K. Popplewell, "The current status of process planning for multi-material rapid prototyping fabrication," *Advanced Materials Research*, vol. 118–120, pp. 625-629, 2010.

[57] S. H. Choi, H. H. Cheung, and W. K. Zhu, "A multi-material virtual prototyping system for biomedical applications," in *IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurements Systems*, Hong Kong, 2009, pp. 73-78.

[58] R. Law, S. Dorfman, C. Khatiwala, and B. Shepherd, "Scaffold free NovoGen bioprinting using various hydrogels," in *TERMIS-NA Annual Meeting*, Orlando, Florida, 2010.

[59] G. Teng, J. Zhifeng, and F. Jianglong, "Research of NC Code Interpreter Based on Theory of Finite Automaton," *Modern Applied Science*, vol. 6, no. 4, pp. 38-43, 2012.

[60] Galil Motion Control Inc., *CAD-to-DMC User Manual Rev. 1.0b*, Galil Motion Control Inc., Rocklin, California, 2006.

[61] S.T. Beyer, A. Bsoul, A. Ahmadi, and K. Walus, "3D alginate constructs for tissue engineering printed using a coaxial flow focusing microfluidic device," in *The 17th International Conference on Solid-State Sensors, Actuators and Microsystems (Transducers & Eurosensors XXVII)*, Barcelona , 2013, pp. 1206-1209.

[62] S. Beyer, T. Mohamed, and K. Walus, "A microfluidic based 3D bioprinter with on-the-fly multimaterial switching capability," in *The 17th International Conference on Miniaturized Systems for Chemistry and Life Sciences (MicroTAS 2013)*, Freiburg, 2013, pp. 176-178.

[63] Galil Motion Control Inc. , *DMC-2x00 Manual Rev. 2.1*, Galil Motion Control Inc., Rocklin, California, 2011.

[64] Galil Motion Control Inc., *Command Reference Manual Rev. 1.0u*, Galil Motion Control Inc., Rocklin, California , 2009.

[65] D. Batory and S. O'malley, "The design and implementation of hierarchical software systems with reusable components," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 1, no. 4, pp. 355-398, 1992.

[66] F. J. Van Der Linden and J. K. Muller, "Creating architectures with building blocks," *IEEE Software*, vol. 12, no. 6, pp. 51-60, 1995.

[67] I. Crnkovic, "Component-based software engineering—new challenges in software development," *Software Focus*, vol. 2, no. 4, pp. 127-133, 2001.

[68] R. Sanchez and J. T. Mahoney, "Modularity, flexibility, and knowledge management in product and organization design," *Strategic management journal*, vol. 17, no. S2, pp. 63-76, 1996.

[69] M. Lombard, *SolidWorks 2013 Bible*. Indianapolis, Indiana: John Wiley & Sons, 2013.

[70] L. Prechelt, "Comparing Java vs. C/C++ Efficiency Differences to Interpersonal Differences," *Communications of the ACM*, vol. 42, no. 10, pp. 109-112, 1999.

[71] J. Levine, *Flex & Bison: Text Processing Tools*. New York: O'Reilly Media Inc., 2009.

[72] D. Brown, J. Levine, and T. Mason, *Lex & Yacc*. New York: O'Reilly Media Inc., 1992.

[73] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, "Extensible Markup Language," *World Wide Web Journal*, vol. 2, no. 4, pp. 29-66, 1997.