Automated Design of Graded Material Transitions for Educational Robotics Applications

by

Cole Brauer

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved April 2020 by the
Graduate Supervisory Committee:

Daniel Aukes, Chair
Xiangfan Chen
Thomas Sugar

ARIZONA STATE UNIVERSITY

May 2020

ABSTRACT

Multi-material fabrication allows for the creation of individual parts composed of several materials with distinct properties, providing opportunities for integrating mechanisms into monolithic components. Components produced in this manner will have material boundaries which may be points of failure. However, the unique capabilities of multi-material fabrication allow for the use of graded material transitions at these boundaries to mitigate the impact of abrupt material property changes.

The goal of this work is to identify methods of creating graded material transitions that can improve the ultimate tensile strength of a multi-material component while maintaining other model properties. Particular focus is given towards transitions that can be produced using low cost manufacturing equipment. This work presents a series of methods for creating graded material transitions which include previously established transition types as well as several novel techniques. Test samples of each transition type were produced using additive manufacturing and their performance was measured. It is shown that some types of transitions can increase the ultimate strength of a part, while others may introduce new stress concentrations that reduce performance. This work then presents a method for adjusting the elastic modulus of a component to which graded material transitions have been added to allow the original design properties to be met.

ACKNOWLEDGMENTS

This research was supported by IDEAlab and the MORE program at Arizona State University. Continued development on the VoxelFuse platform to facilitate this work is supported by Jude Brauer and Charles Jeffries.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Multi-material fabrication describes a family of fabrication methods that can be used to produce individual parts that can be made up of several different materials. It provides a way to create single components that can take the place of assemblies of components produced using conventional techniques. This allows for robot designs that are both robust and low cost, making it a particularly attractive method for education or research. Example applications in robotics include integrated flexible joints between rigid materials, soft grip surfaces built into parts, or embedded reinforcing materials in components. Multi-material fabrication can be directly realized through some additive manufacturing techniques. Examples include resin-based 3D printing processes that can achieve materials with varying hardness, or Fused Deposition Modeling (FDM) processes on multi-head 3D printers that can deposit 2 or more discrete materials. Multi-material fabrication can also be accomplished by combining multiple fabrication techniques. One example is producing rigid regions using 3D printing then adding flexible regions using a resin casting process.

Using multi-material fabrication introduces material boundaries into a part that may be points of failure. This is typically addressed through the use of ”dog bone” joints which provide a mechanical connection to supplement adhesion between materials. While effective, this method requires a relatively large volume of material around the joint to be dedicated to the connecting structure and does not generalize well to work with any boundary surface. It also leaves a distinct change in material properties which may be a stress concentration. Another method that can enhance material transitions is the use of functionally graded materials. These use a gradient in the distribution of materials throughout a part to provide a smooth transition in mechanical properties between different regions of the part. This method is easily generalized to any boundary surface, but it can greatly limit the types of materials and processes that can be used.

The goal of this work is to identify methods of creating graded material transitions that can improve the ultimate tensile strength of a multi-material component while maintaining other model properties. Particular focus is given towards transitions that can be produced using low cost manufacturing equipment. Chapter 2 describes the state of the art in functionally graded material transitions and introduces the software framework used for processing multi-material models. Chapter 3 describes the further development of this framework that was performed to support the generation of graded material transitions. Chapter 4 presents a series of methods for translating a true graded material transition into different structures that can be made using two discrete materials to approximate the target material distribution. These include previously established transition methods as well as several novel techniques. Chapter 5 describes the procedures used to generate, fabricate, and test these structures. Chapter 6 presents the results of this testing and compares them to other model properties. Chapter 7 summarizes the contributions of this work and discusses areas for further research. The transition generation code has also been provided to allow it to be easily applied in future work.

Chapter 2

BACKGROUND

Previous work has explored the use of embedded flexible materials in robotics applications. Examples include shock-absorbing joints in robot legs (Cham *et al.*, 2002), joints and grips for robotic hands (Ma *et al.*, 2013), and structures with a specific deformation profile (Hiller and Lipson, 2012). These projects show that multi-material structures can provide versatile, robust, and low cost methods for robot construction.

Previous work illustrates two methods for producing strong joints between materials. The robotic hand presented by Ma *et al.* uses dog bone joints to connect 3D printed plastic components and cast rubber components. This type of connection is easily produced, but it is relatively large and limits the geometry of the joint. Another method is embedding fiber reinforcement into linkages made with hard and soft cast resins (Hatanaka and Cutkosky, 2003). This method is strong and compact, but it requires a manual insertion step that does not translate well to non-casting processes and that is difficult to automate. Both techniques still leave a distinct change in material properties.

One approach that has been explored for improving the performance of material transitions is applying a blur to the boundary between two materials (Kaweesa and Meisel, 2018b). This method was shown to provide opportunities for increasing the fatigue life of test specimens. To create the test specimens with a blur between materials, the researchers used a multi-material 3D printer and produced approximations of a blur using both a series of discrete materials and a fine dithering algorithm. These methods are limited to printers with multi-material capabilities and fine print resolution.

The impact of voxel size on the tensile strength of dithered test samples has been previously researched (Kaweesa and Meisel, 2018a). The researchers found that increasing voxel size will reduce tensile strength and shift elongation properties towards the dominant material, but that it also provides a way to reduce computation time. It demonstrates that when using voxel-based structures, the trade off between computation time and material property accuracy must be considered.

Another method that has been proposed for creating functionally graded material transitions is the use of triply periodic minimal surfaces to create two-material parts that are mechanically interlocked (Stoner *et al.*, 2018). Graded properties in the transition were achieved by varying the thickness or period of the structure. This approach was tested as an interface between two parts of the same material and was found to significantly improve the tensile strength of the test specimen compared to a specimen with a binary material interface. This approach can be easily adapted to apply to parts containing two materials with distinct properties.

One method for creating single material structures with graded properties that is not material-specific is the use of lattice structures with varying member thickness (Aremu *et al.*, 2017). Depending on the lattice element design used, these structures can be easily produced with many different additive manufacturing methods. By combining a lattice structure and its negative model, this method can be applied to creating graded material transitions.

The interface between muscles and bones can also provide inspiration for creating strong material transitions. This interface consists of many small fibers that unravel from the muscle then slowly transition to a more rigid material (Rossetti *et al.*, 2017). This concept can be simplified and scaled up to create a printable structure.

Previous work by the author of this paper introduced a new framework and an accompanying software tool, VoxelFuse, for planning functionally graded and multi-step fabrication processes (Brauer and Aukes, 2019; Brauer *et al.*, 2020). The framework includes a voxel-based representation for multi-material models, low-level operations for combining geometries, and algorithms for assisting a designer in computing manufacturing-compatible sequences. A simple application that was demonstrated was the generation of a graded material transition using a Gaussian blur. For this work, VoxelFuse was used as the foundation for scripts to automate the creation of different transition types.

This work provides a comprehensive comparison of different graded material transition types. It includes the previously developed transition methods discussed above based on blurring and triply periodic minimal surfaces. It also includes methods based on macro-scale dithering, lattice structures, and macro-scale fibers that to our knowledge have not previously been applied to graded material transitions.

Chapter 3

VOXELFUSE FRAMEWORK

In order to generate graded material transitions, a system was needed for representing and processing 3D models that include material data. For this project, the VoxelFuse platform was used (Brauer *et al.*, 2020). Previously developed functionality of this platform included performing basic constructive solid geometry, material, morphology, blurring, and transformation operations as discussed in (Brauer and Aukes, 2019). In order to generate and produce the graded transition types required by this project, new capabilities were required for generating dithered structures, primitive solids, and triply periodic minimal surfaces. This chapter summarizes the material representation used by VoxelFuse and details these improvements.

Model Representation

VoxelFuse uses a voxel-based representation to store and process multi-material models. Each voxel contains a vector representing the materials present at that voxel. The material vector for a specific voxel in the model $\boldsymbol{V}$ is defined by

$$\boldsymbol{V}_{\langle i,j,k \rangle} = \langle a, m_0, ..., m_n \rangle, \tag{3.1}$$

where a is a boolean value indicating the presence or absence of material at the corresponding voxel, and $m_{0...n}$ are an arbitrary number of material channels that contain floating point values representing the percentages of materials present. For example, a voxel with $a = 1$, $m_1 = 0.8$, $m_2 = 0.2$ will contain 80% material 1, and 20% material 2. The m_0 value represents a null material and is used to express voxels containing a material density of less than 100%. For example, a voxel with $a = 1$,

$m_0 = 0.5$, $m_1 = 0.5$ will contain material 1 at 50% density. Similar approaches are also discussed in (Kou and Tan, 2007; Bader, 2017).

By storing material data in this manner, varying mixtures and densities of materials can be easily represented throughout a model. This is also a similar structure to that of a color image, allowing many image processing algorithms to be adapted to operate on 3D data. In addition to the Gaussian blurring algorithm already developed for VoxelFuse, one image processing algorithm that is of particular interest for material interface generation is dithering. The development of a 3D dithering function is discussed further in the following section.

Dithering

Dithering provides a method of simplifying a region of a model containing a continuum of values, such as the result from a blurring operation, into a structure composed of a small number of discrete values. It is of interest in the generation of material transitions because it can allow for graded properties to be approximated using only two materials.

To perform dithering on a multi-material 3D model, the following process is performed sequentially for each voxel in the model. First, the voxel's material is set to the single material that is present in the highest percentage at that voxel. The error caused by changing the material is then calculated by subtracting the new material percentages from the original values. The error values for each material channel are then distributed to the surrounding voxels based on a diffusion filter. For this work, a 3D extension of a Stucki filter was used as discussed in Section 4 of (Lou and Stucki, 1998). This filter is illustrated in Figure 3.1, with X representing the voxel that was just updated. Lou and Stucki note that some error diffusion filters, including the Stucki filter, can be extended to support three dimensions while others, such as the Floyd-Steinberg filter, cannot because they will introduce artifacts along certain planes.

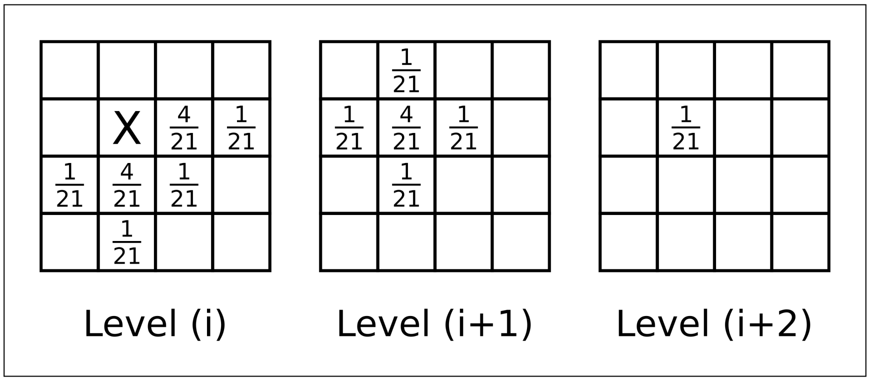


Figure 3.1: 3D Stucki Error Diffusion Filter

When implementing this process as a VoxelFuse function, an additional thresholding step was added to prevent error from being distributed to voxels with a high enough percentage of one material. This prevents small amounts of error from accumulating and changing the material of voxels outside of the transition region.

The application of a dithering to material transitions is discussed further in Section 4. The code developed for performing 3D dithering is included in Appendix A.

Model Generation Functions

Most material transition designs require the generation of new model elements. Examples range from generating a new rectangular gauge section to generating a periodic surface in a transition region. For this project, two new sets of functions were defined and added to VoxelFuse: primitive solids and periodic structures. These functions are now included in the main VoxelFuse repository (Brauer *et al.*, 2020).

Primitive Solids

Primitive generation functions create basic solid shapes based on the desired dimensions. This class of functions was used for generating additions to models as well as volumes that could be intersected with a model to isolate specific regions. Algorithm 1 provides a general definition of a primitive generation function. In this function, *size* represents one or more values which define the size of the output and *material* represents the desired material vector for the shape. It is assumed that the material distribution is uniform upon generation. EmptyModel() is a function of *size* that initializes a new model array to hold the generated primitive. The condition(s) to determine if a specific voxel is contained in the volume of a shape will depend on the specific shape being generated. For example, to generate a sphere, the distance from the center of the model to a voxel would be compared to the target radius of the sphere as defined by *size*.

Algorithm 1 Primitive Solid Generation

function Shape($size, material$)
  $\boldsymbol{V} \leftarrow$ EmptyModel($size$) ▷ Initialize new empty model array
  for all $\langle x, y, z\rangle \in \boldsymbol{V}$ **do** ▷ For all coordinates
    if $\langle x, y, z\rangle \in$ (Volume of Shape) **then** ▷ If voxel is contained in shape
      $\boldsymbol{V}_{\langle x,y,z\rangle} \leftarrow material$ ▷ Fill voxel in new model
    end if
  end for
  return $\boldsymbol{V}$
end function

Primitive generation functions were created for solids including cubes, cuboids, spheres, cylinders, cones, and pyramids. These are illustrated in Figure 3.2.

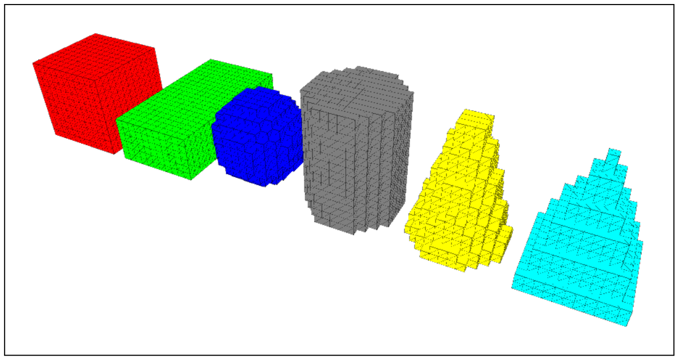


Figure 3.2: Results Created by Primitive Generation Functions

Periodic Structures

One potential method for creating a graded material transition is to use a triply periodic minimal surface to find two interlocking components. In this use case the surface itself is not desired, but rather the two models resulting from cutting a volume with the periodic surface. The process for generating two interlocking periodic elements is defined in Algorithm 2. This function operates in a similar manner to the function for generating shapes. Rather than checking if a voxel is within a target shape, this function checks whether the voxel is above or below a target periodic surface.

Functions for generating periodic structures over cuboid regions were created for gyroid surfaces, p-surfaces, and d-surfaces. An example output for a structure created using a gyroid surface is shown in Figure 3.3. The the periodic function $F(x, y, z)$ used for each type of surface and the application of a periodic structure to a material transition are discussed further in Section 4.

Algorithm 2 Periodic Element Generation

function PERIODIC($size, material$)
  $\boldsymbol{P}, \boldsymbol{N} \leftarrow$ EMPTYMODEL($size$) ▷ Initialize two empty model arrays
  for all $\langle x, y, z \rangle \in \boldsymbol{P}$ **do** ▷ For all coordinates
    if $\mathrm{F}(x, y, z) > 0$ **then** ▷ If periodic function is positive
      $\boldsymbol{P}_{\langle x,y,z \rangle} \leftarrow material$ ▷ Fill voxel in positive model
    else
      $\boldsymbol{N}_{\langle x,y,z \rangle} \leftarrow material$ ▷ Fill voxel in negative model
    end if
  end for
  return $\boldsymbol{P}, \boldsymbol{N}$
end function

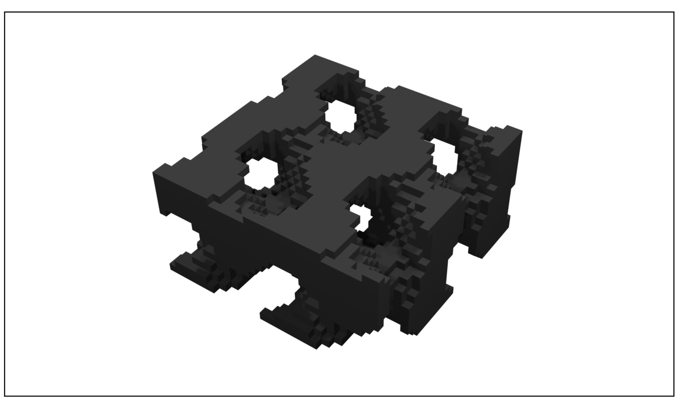


Figure 3.3: Gyroid Structure Generated Over a 30x30x15 Voxel Region

Chapter 4

TRANSITION TYPES

Based on the background research discussed in Section 2, several methods for generating transitions with graded properties were identified. These types of transitions are as follows:

1. Blur
2. Dither
3. Lattice structure
4. Triply periodic minimal surface
5. Fibers

Many of these transitions also provide a potential benefit by increasing the contact surface area or providing mechanical connection between the two joined materials. The sections below discuss each type in detail. The sample model used is based on the ASTM D638 standard (ASTM-D638-14, 2014) and is shown in Figure 4.1. This model is discussed further in Section 5. For layer-based processes, it is assumed that layers are deposited in the XY plane and build in the positive Z direction.

Blur

A blurred material transition can be computed by applying a Gaussian blurring algorithm to each material channel in a model in the region surrounding a material boundary. The resulting blur will be centered about the original material boundary.

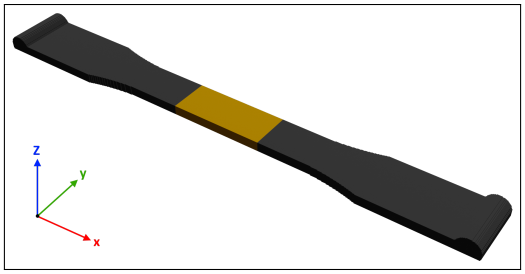


Figure 4.1: Sample Model with No Transition Applied

This process is illustrated in Figure 4.2, and the result of applying it to the sample model is shown in Figure 4.3.

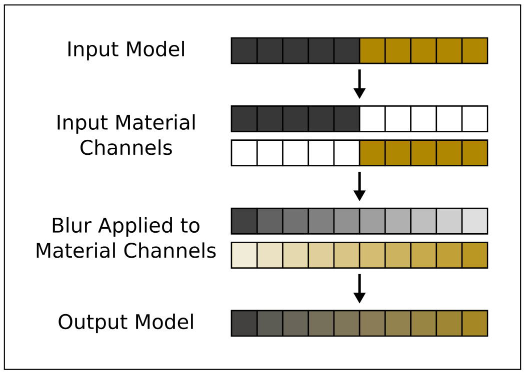


Figure 4.2: Blurring Process

A blur represents a true graded material transition with no boundaries between materials. It is easily applied to any material boundary geometry, including those in thin model regions where a mechanical connection such as a dog bone is difficult to fit.

To produce a blurred transition, a manufacturing process that supports multiple materials and that has a high resolution is required. Even with this type of process, the blur must be approximated by narrow bands of discrete material, as shown in Figure 4.3, or by a micro-scale dithered pattern. The remaining types of transitions represent methods of translating a blurred transition into a structure that is more

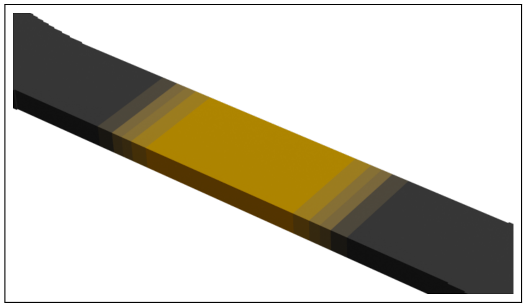


Figure 4.3: Sample Model with Gaussian Blurring Applied

easily produced.

Dither

The most basic simplification of a blurred material transition is applying a dithering algorithm to approximate the material distribution using two materials. To ensure support for any boundary surface orientation, a 3D dithering algorithm is needed. The development of a 3D dithering function is discussed in Section 3. The result of applying a dithering algorithm to the blurred model from Section 4 is illustrated in Figure 4.4.

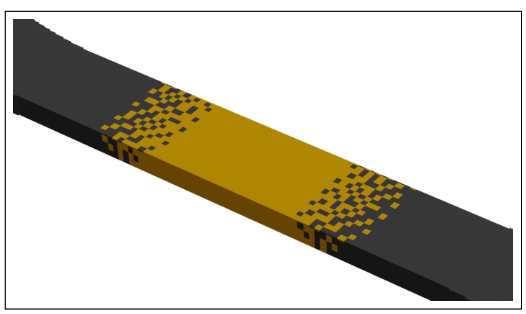


Figure 4.4: Sample Model with 3D Dithering Applied

This approach can be scaled to the resolution capabilities of a given process by applying a macro-scale dither with a voxel resolution matching the desired minimum feature size. As noted in (Kaweesa and Meisel, 2018a), large voxel sizes in a dither can shift the material properties away from the target properties, so the resolution should be kept to the minimum size possible in order to provide the best approximation of a blur.

The result from applying a dithering algorithm is compatible with 3D printing processes capable of depositing two or more materials. If two separate manufacturing processes are to be used for the two materials, an additional constraint will be placed on the dithering in that all disconnected components must be fully supported along the Z-axis with respect to the ground plane. By adjusting the error diffusion filter to remove error propagation in Z, a model with the dithered pattern repeated along the Z-axis can be achieved. This is illustrated in Figure 4.5. It should be noted that this will affect the accuracy of the dithered result with respect to the blurred model, particularly when the material boundary plane does not contain the Z-axis. In addition, each layer of a dithered model will still contain a number of disconnected components. While this does not necessarily affect manufacturability, it may increase production time or difficulty depending on the types of processes used.

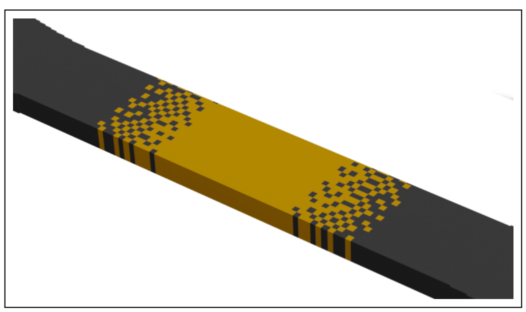


Figure 4.5: Sample Model with 2D Dithering Applied to Maintain Support Along the Z-Axis

A dithered transition benefits from a gradual change in material properties, but it provides little to no mechanical connection.

Lattice Structure

A graded lattice structure for one material in a transition can be found by using the blurred model to determine the desired density of each lattice element in a grid of elements that encompasses the model. The lattice elements with the desired densities can be generated by applying dilate or erode operations to a template element. The mating structure can then be found by subtracting the lattice structure from the original model. An example of this process is shown in Figure 4.6.

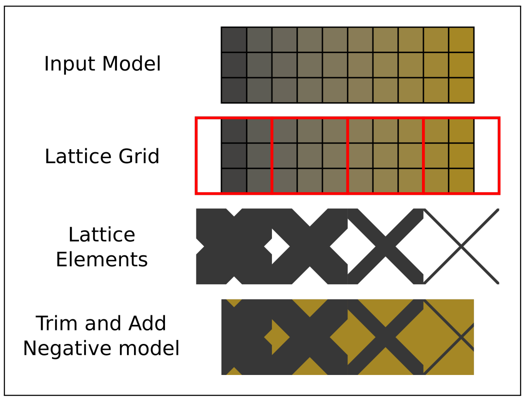
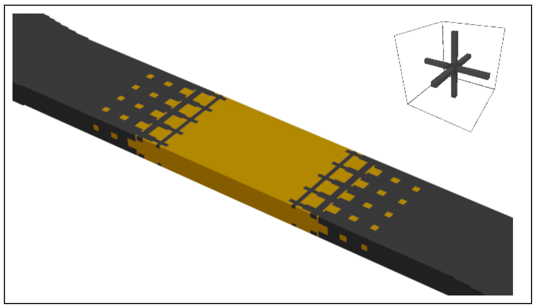


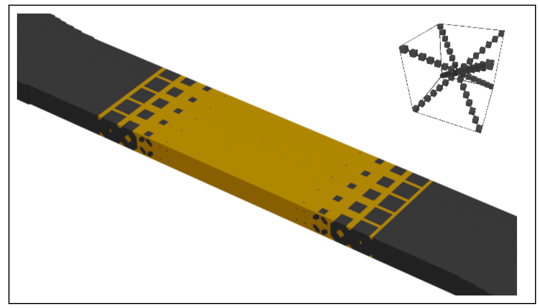
Figure 4.6: Process for Generating a Graded Transition Using a Lattice Structure

Several factors must be considered when designing the lattice element template. The element must tessellate in three dimensions while connecting to its neighbors. If separate processes are used for the lattice and its negative, element overhangs may need to be designed so that they are supported in Z or can produced by bridging two regions that are supported in Z. If using a layer-based process, it may be desirable to choose a lattice design that possesses layers with no disconnected components in adjacent elements. Finally, if using a casting-based process, the lattice and/or its negative should consist of a single continuous component so that only a single pour location is needed.

Two example lattice element patterns and the resulting graded transitions are shown in Figure 4.7. Structure (a) can be produced without supports via bridging, while structure (b) is supported in Z via angled overhangs. Structure (a) also contains layers in the XY plane with no disconnected components. Both structures have a negative that is a continuous component.



(a)



(b)

Figure 4.7: Lattice Structure Transitions Using (a) a Plus-Shaped Lattice Element and (b) an X-Shaped Lattice Element

A lattice structure generated in this manner can be applied to any boundary surface orientation. However, if the lattice elements are intended to be in a specific orientation relative to the boundary plane, an additional step may be required to rotate each element.

Due to the relatively large size of individual lattice elements, this approach will have larger steps in material properties compared to a stepped blur. It will also have higher space requirements than a blur or dithering approach. However, a lattice structure is easier to produce and will provide a stronger mechanical bond.

Triply Periodic Minimal Surface

A triply periodic minimal surface provides a way to divide a volume into two interlocking components with the same shape. This type of surface is defined by a periodic function of X, Y, and Z, such as the examples shown in Equations 4.1-4.3.

$$\text{Gyroid} \quad \begin{aligned} &sin(xs) * cos(ys) + \\ &sin(ys) * cos(zs) + \\ &sin(zs) * cos(xs) = 0 \end{aligned} \tag{4.1}$$

$$\text{P Surface} \quad cos(xs) + cos(ys) + cos(zs) = 0 \tag{4.2}$$

$$\text{D Surface} \quad \begin{aligned} &sin(xs) * sin(ys) * sin(zs) + \\ &sin(xs) * cos(ys) * cos(zs) + \\ &cos(xs) * sin(ys) * cos(zs) + \\ &cos(xs) * cos(ys) * sin(zs) = 0 \end{aligned} \tag{4.3}$$

In these equations, s represents a scaling factor and is equal to 2π/(voxels per period). By using such a surface to split a cube with dimensions equal to the period of the function, a two-material element can be found which can be used to generate a graded transition in the same manner as a lattice structure. By applying dilate or erode operations to each material, the material proportions can be made to vary throughout the model. Figure 4.8 shows three graded transitions generated using Equations 4.1-4.3 and the algorithm described in Section 3.

A material transition based on a triply periodic minimal surface shares many of the characteristics of a transition based on a lattice structure. It has benefits over a lattice structure in that the element design is defined based on a function and as such does not need to be manually modeled and and is easily adjusted. Like a lattice structure, support in Z can be accomplished using angled overhangs or bridging and some functions will perform better in this area than others.

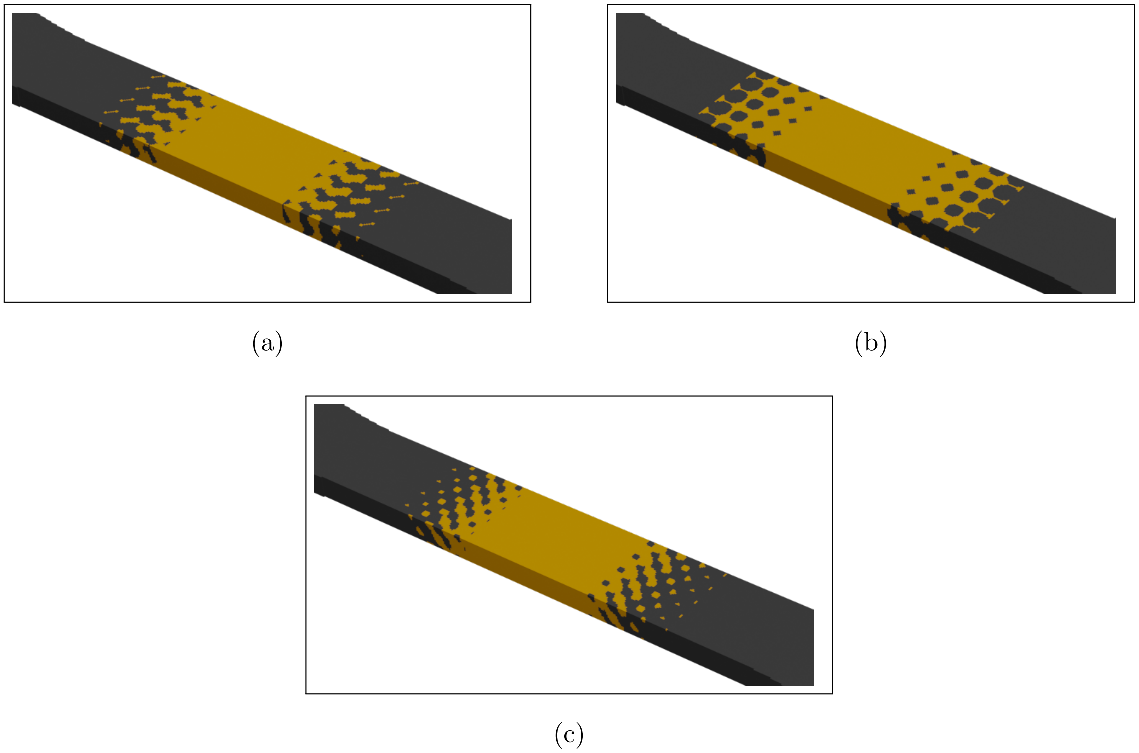
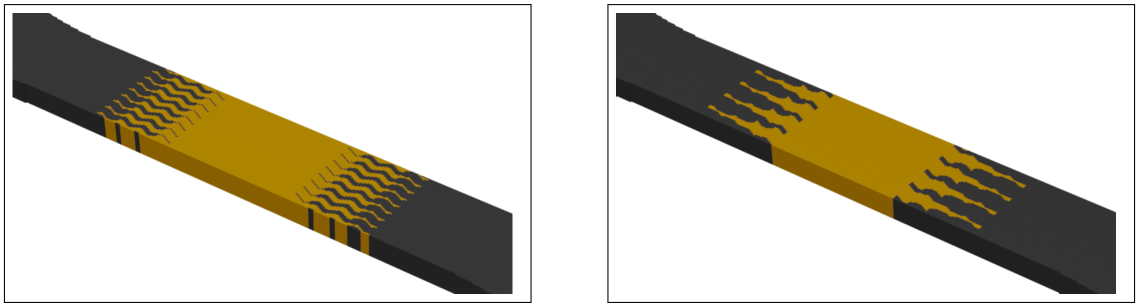


Figure 4.8: Graded Transitions Defined Using (a) a Gyroid Surface, (b) a P-Surface, and (c) a D-Surface

Fibers

The connection between bones and muscles is one example of a strong material transition in nature. This connection uses micro-scale fibers to improve the bond strength between two dissimilar materials (Rossetti *et al.*, 2017). By taking inspiration from this structure, a material transition based on macro-scale fibers can be designed. An example design in two different scales is shown in Figure 4.9.

This structure can be implemented by creating a template element that can be applied to a blur in the same manner as a lattice structure. By adjusting the base thickness and spacing of the fibers, this method can be made compatible with processes with different minimum feature sizes. Unlike the previous two methods, the



(a) (b)

Figure 4.9: Fiber Transitions Using (a) Small and (b) Large Template Elements

pattern elements must be in a specific orientation relative to the material transition boundary. This can be accomplished by rotating the template elements to align with a line normal to the boundary plane.

This type of transition provides a large increase in contact surface area as well as some mechanical bonding. It also does not have any layers with disconnected components, which can simplify manufacturing.

Chapter 5

EXPERIMENTAL SETUP

For this work, the primary measure of performance used for material transitions is their impact on the ultimate tensile strength of a part. To measure this, a series of tensile test coupons was created representing each transition type discussed in Section 4. These coupons were produced via 3D printing, and tensile testing was used to measure the performance of each transition type. Based on the results of testing the first set of coupons, a revised set was created and tested to investigate how tensile strength could be improved without modifying the part's modulus of elasticity.

Test Coupon Design

The base coupon design was created in accordance with the ASTM D638 Type I specification (ASTM-D638-14, 2014). This specification applies to testing the tensile properties of rigid and semi-rigid plastics. A region was added to the center of the gauge section to represent the secondary material type. The boundary planes between this section and the end sections are oriented perpendicular to the axis of loading. For this work, a rigid material is used for the ends of the sample while a flexible material is used for the center region of the sample. A model of the coupon was created using Autodesk Fusion 360 (Autodesk Inc., 2020), then exported in an STL format. This model is shown in Figure 5.1.

The gauge section of this coupon has a width (W) of 13 mm and a thickness of 3.2 mm. The length of the secondary material region (L_0) is 31 mm. Transitions are

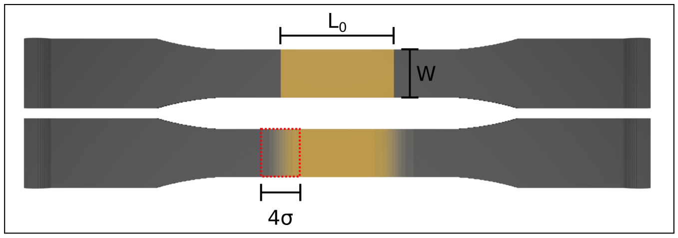


Figure 5.1: Input Model (Top) and Output Model After Applying a Material Transition (Bottom)

applied to this model based on a Gaussian blur with a standard deviation σ, resulting in a transition region that is approximately 4σ long. Circular profiles with a diameter of 8 mm were added to each end of the coupon to aid in clamping samples in a tensile testing machine.

Transition Generation

The transition types represented in the set of test coupons are listed in Table 5.1. The binary interface represents no transition modification and serves as a performance baseline. Transition models were generated with a Python script created using the VoxelFuse library. This script is included in Appendix B.

The coupon STL file was imported into the VoxelFuse script using a linear resolution of 5 voxels / mm. This value was selected for being able to adequately represent the resolution capabilities of low cost printing processes without posing a barrier in computation time. The initial blur for all transitions uses a standard deviation (σ) of 3 mm to yield a transition region that is 12 mm long.

For blurred and dithered transitions, the model was scaled down to a linear resolution of 1 voxel / 4 voxels, the transition was applied, then the model was scaled back to its original size. This results in a minimum feature size of 0.8 mm, which is compatible with common 3D printing technologies.

Label	Design
A	Binary
B	Blur
C	3D dither
D	2D dither
E	Gyroid
F	P-surface
G	D-surface
H	+ lattice
I	X lattice
J	Small fibers
K	Large fibers

Table 5.1: Tensile Test Coupon Types

For triply periodic surfaces, a 15 voxels cubic element was generated which contained a single period of the surface. This element size results in approximately one layer of elements in a 3.2 mm thick coupon. The element was then applied to the model using the procedure described in Section 4.

For lattice structures and fibers, a template element with a size of 15 voxels was first created using MagicaVoxel (Ephtracy, 2018). The template element was then imported and applied to the model using the procedure described in Section 4.

After the coupon models were generated, they were combined into a single model and distinct materials were exported as a series of STL files. These files can be imported into a 3D printer's control software and the print settings for each material can be set independently.

Equipment Setup

To produce and test the sample models, a multi-material 3D printer and a tensile testing system were used. Details on the equipment selected and the setup for each are discussed below.

3D Printer

All test specimens were produced using a Stratasys Objet 350 3D printer. This printer uses the PolyJet printing process, which simultaneously deposits both rigid and flexible resin materials. By mixing these materials, it can produce rubber-like materials in a range of hardness values between Shore 30A and 95A (Stratasys, 2016). The Objet 350 is capable of producing all of the transition types described above, including the stepped blur. For this work, RGD5130-DM and FLX95595-DM material cartridges were used to create the rigid and flexible coupon materials respectively. For two material patterns, the flexible portion of the model was set to a hardness value of Shore 60A. This value was selected such that the test samples would fail in the transition region for the majority of the samples. For the blur, three additional steps of material at Shore 70A, 85A, and 90A were used.

For testing done up to this point, each set of 11 coupons was printed in the same location in the print bed with the samples arranged in parallel. Future testing will include printing additional samples in other print bed locations and other orientations. This data will help mitigate any effects caused by printer setup.

Future testing will also make use of an Ultimaker S5 3D printer. This printer is representative of low-cost options for multi-material 3D printing that are readily available for educational settings. It uses two extruder heads to alternately deposit two discrete materials. Samples produced with this printer will use Nylon and TPU filament as the rigid and soft materials. Since this printer cannot mix materials, it cannot produce the blurred sample. It is also expected to have reduced bonding strength between the two materials.

Tensile Testing System

The printed models were tested using an Instron 8801 fatigue testing system. Samples were secured using textured plastic adapters to prevent them from slipping in the clamps. The test setup is shown in Figure 5.2. All samples were pulled until failure at a rate of 5 mm/min. Position and load data during the test were collected at a rate of 100 Hz.

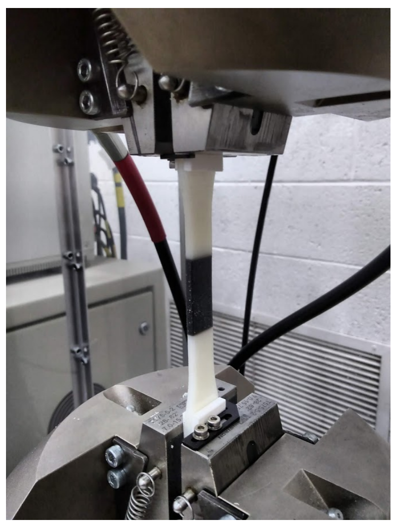


Figure 5.2: Test Sample Mounted in Tensile Testing Machine

Testing Configurations

Two stages of tensile tests were performed. In each stage, three identical sets of coupons printed on the Objet 350 were used.

Stage 1: Centered Material Transitions

In the first stage of testing, coupons were produced with the transition regions generated centered on the two material boundaries. This approach maintains the overall ratio of rigid to flexible plastic. However, it in effect reduces the value of L_0, resulting in a change to the overall modulus of elasticity for the sample. Depending on the specific transition pattern, this effect will be more or less pronounced. The complete set of printed samples is shown in Figure 5.3. The results from this stage of testing are documented in Section 6.

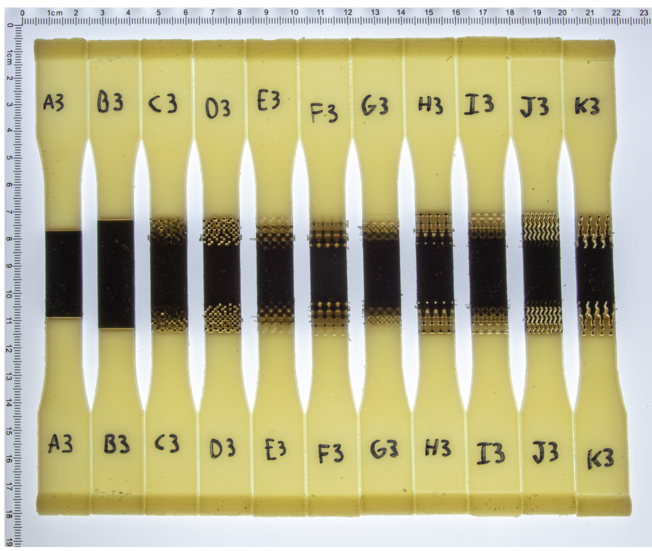


Figure 5.3: 3D Printed Test Samples with Centered Transition Regions

Stage 2: Normalized Modulus of Elasticity

In some applications, it may be desired to increase the ultimate tensile strength of a part without modifying its modulus of elasticity. One such application would be if the part includes a flexible region that acts as an integrated spring. In this case, a method is needed for adjusting the material transitions to compensate for the reduction in the effective length of L_0.

Using the results of the first set of tests, the effective value of L_0 was found for each sample. A new set of coupons with a normalized modulus of elasticity was then generated by using these values to scale the value of L_0 for each model before generating material transitions. Only the length of the transition region was scaled so as to maintain the overall design geometry. The complete set of printed samples is shown in Figure 5.4. The amount of adjustment needed for each pattern and the results from this stage of testing are discussed further in Section 6.

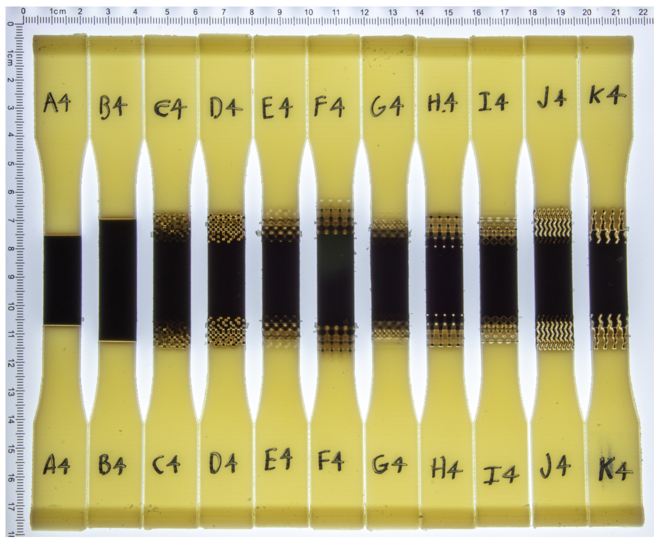


Figure 5.4: 3D Printed Test Samples with Adjusted Transition Region Positions

Future Testing

The same two stages of testing will also be performed using samples printed with the Ultimaker S5. For the first stage of testing with the Ultimaker the same model files will be used as in Section 5, with the exception that the blurred transition will not be printed. Other than the blur, all of the transition designs can be produced on a two-material FDM 3D printer. However, designs that have layers with a large number of disconnected components, such as the dithered patterns or the lattice structures, may have a long production time with this type of process. The results from this testing these samples will be used to generate a new set of normalized coupons for the Ultimaker. It is expected that the scaling factors will differ from those for the Objet due to changes in material properties and adhesion strength.

Chapter 6

RESULTS AND ANALYSIS

After testing the coupons, the data saved by the tensile testing system was filtered to reduce sensor noise and offset. The stress-strain curve, ultimate tensile strength, and modulus of elasticity for each sample were then plotted and compared. The point of failure for each sample was visually inspected. In addition, the interface surface area and volume percentage of flexible material for each pattern were computed. The results from each of these tests are discussed below.

Data Filtering Process

The raw data saved by the tensile testing system is in the format of load and displacement. The sensor offset for each test session was found by running a test cycle with no sample loaded and averaging the resulting load signal. The raw test data was first updated with these offset values, then converted to stress and strain values based on the design gauge length and cross section. A low pass filter with a cutoff frequency of 1 Hz was applied to remove noise. The three trials for each coupon design were averaged to obtain a single signal for each. The failure point of each design was identified by a sharp drop in the stress value between two data points. Finally, the standard deviations of the three stress values for each failure point were found. All processing and plotting of data was done using MATLAB.

Stage 1 Results

The results from testing the sets of coupons with the transition region centered at the material boundary are shown in Figure 6.1. The error bars on the graph represent the standard deviation of the three trials for each design. Several of the transitions exhibit a significant increase in ultimate stress compared to the binary interface (A), namely the blur (B), 3D dither (C), and 2D dither (D). The ultimate stress of the remaining transitions is equivalent to or less than the binary interface, and reduces in value as the transition moves further away from a true blur.

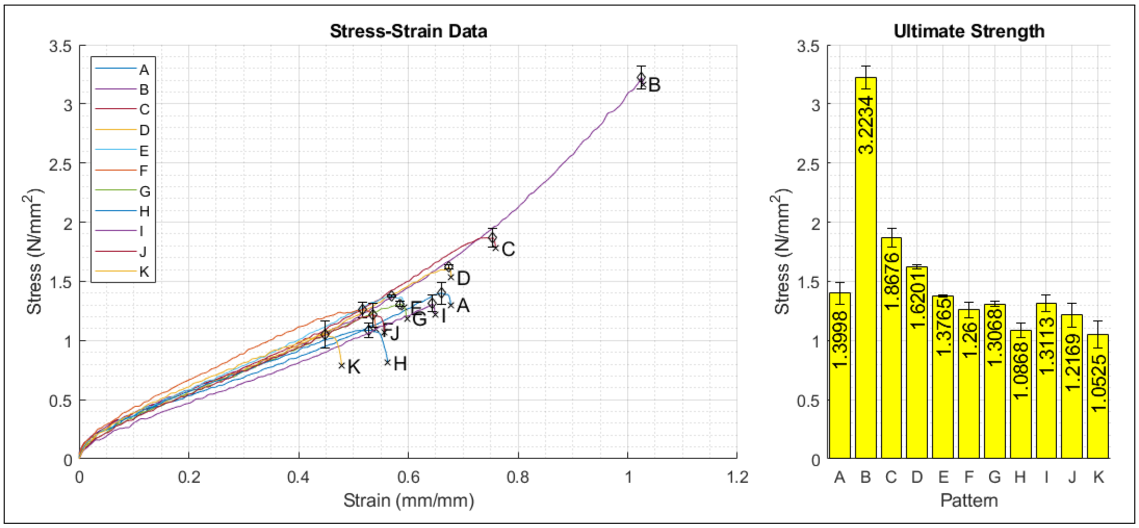


Figure 6.1: Tensile Testing Data for Samples with Centered Transition Regions

The failure points for each design are shown in Figure 6.2. The binary interface failed at the material boundary, the blur failed in the flexible material region, and the remaining designs failed in the transition region.

The modulus of elasticity for each sample was determined by measuring the slope of the stress-strain curve between the strain values of 0.15 and 0.4. It was observed that for most samples, the modulus of elasticity was greater than that of the original binary interface. This is result of the reduced value of L_0 caused by adding transitions. By assuming that the original sample (A) had the intended modulus of elasticity, the effective length of L_0 for each sample was calculated. Table 6.1 summarizes the effective L_0 values calculated from the testing results. The reciprocals of the multiplier

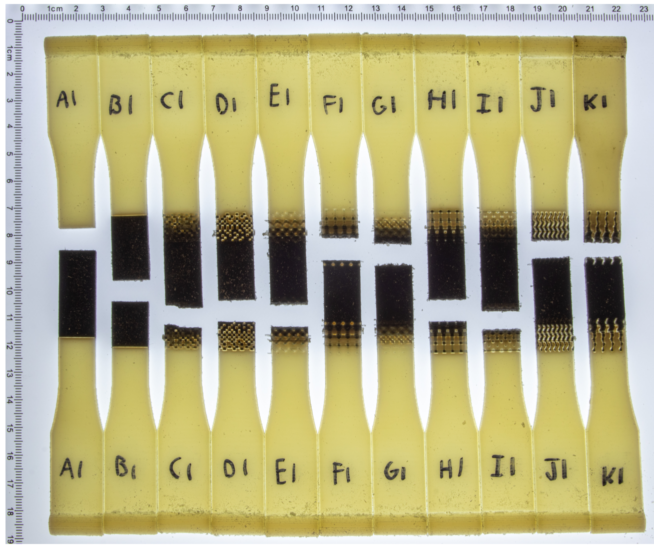


Figure 6.2: Failure Points for Samples with Centered Transition Regions

values represent the amounts that the flexible material regions should be scaled to obtain samples with a normalized modulus of elasticity.

Stage 2 Results

The results from testing the normalized sets of coupons are shown in Figure 6.3. With the exception of the blur (B), the values for the ultimate stress are lower but follow the same trends compared to the first stage of testing. The ultimate stress value for the blurred sample is much lower than in the previous result, putting it below the binary transition in performance. The reduction in ultimate tensile strength is also illustrated in Section 6.

The failure points for each design are shown in Figure 6.4. Again, the binary interface failed at the material boundary and patterns C-K failed in the transition region. However, the blur in this stage of testing also failed in the transition region.

The modulus of elasticity and effective L_0 for each sample were found in the same manner as before. Table 6.2 summarizes these values, and 6.5 compares the

	Design	L_0 Multiplier	Effective L_0
A	Binary	1	31 mm
B	Blur	0.9024	27.9739 mm
C	3D dither	0.8406	26.0575 mm
D	2D dither	0.8610	26.6900 mm
E	Gyroid	0.8247	25.5658 mm
F	P-surface	0.7116	22.0593 mm
G	D-surface	0.8626	26.7415 mm
H	+ lattice	0.9258	28.7001 mm
I	X lattice	0.9930	30.7845 mm
J	Small fibers	0.8390	26.0369 mm
K	Large fibers	0.8464	26.2380 mm

Table 6.1: Effective Length Multiplier for Each Transition Design

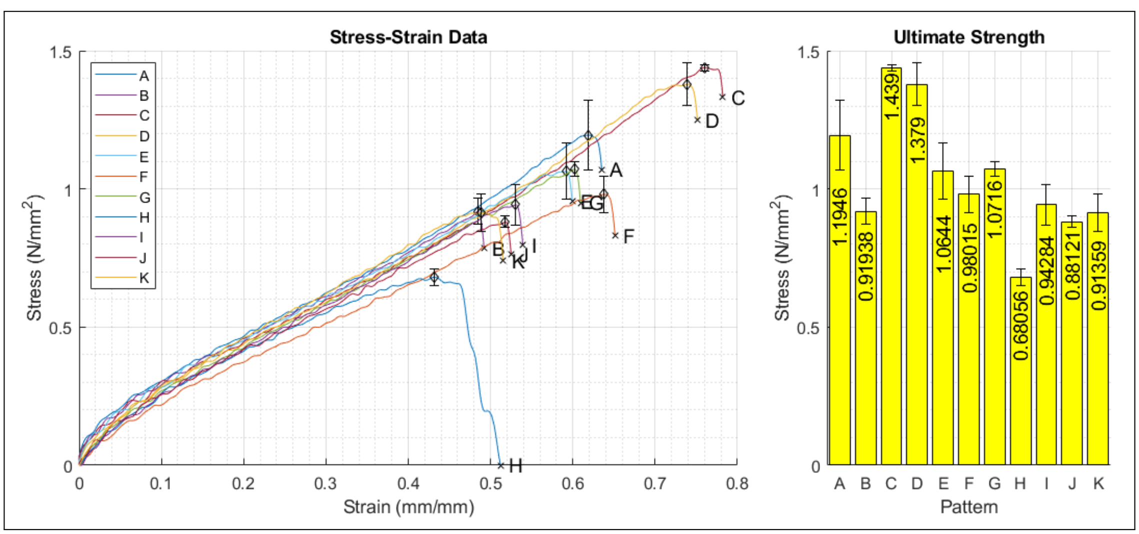


Figure 6.3: Tensile Testing Data for Samples with Adjusted Transition Region Positions

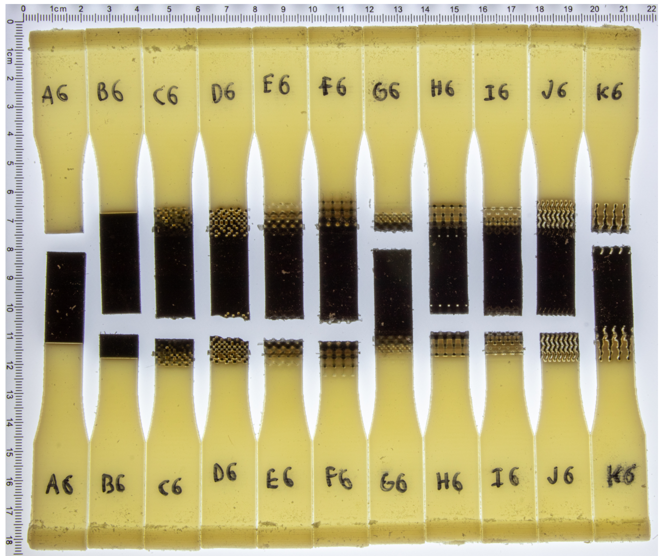


Figure 6.4: Failure Points for Samples with Adjusted Transition Region Positions

distribution of the two sets of multipliers. These values show that the normalized coupons exhibit a modulus of elasticity that is significantly closer to the desired behavior.

Transition Surface Area

Two approaches were identified for determining the contact surface area of a transition. The first is to simply measure the area of all faces where two dissimilar materials meet. The second is to isolate the largest body composed of each material and then measure the measure the contact area between these bodies. This second approach will ignore surface area that is contributed by small, disconnected components. The surface area identified by each method is illustrated by the red border in Figure 6.6. Algorithms to find these two values for contact surface area were implemented using VoxelFuse and applied to the generated transition models. The script for computing these contact surface area values is included in Appendix C. The results for each transition type are illustrated in Figure 6.7.

	Design	L_0 Multiplier	Effective L_0
A	Binary	1	31 mm
B	Blur	1.0234	31.7256 mm
C	3D dither	1.0330	32.0241 mm
D	2D dither	1.0096	31.2975 mm
E	Gyroid	0.9975	30.9216 mm
F	P-surface	1.1405	35.3566 mm
G	D-surface	1.0060	31.1847 mm
H	+ lattice	1.1901	36.8945 mm
I	X lattice	0.9985	30.9537 mm
J	Small fibers	1.0323	32.0011 mm
K	Large fibers	0.9173	28.4356 mm

Table 6.2: Effective Length Multiplier for Each Transition Design After Normalization

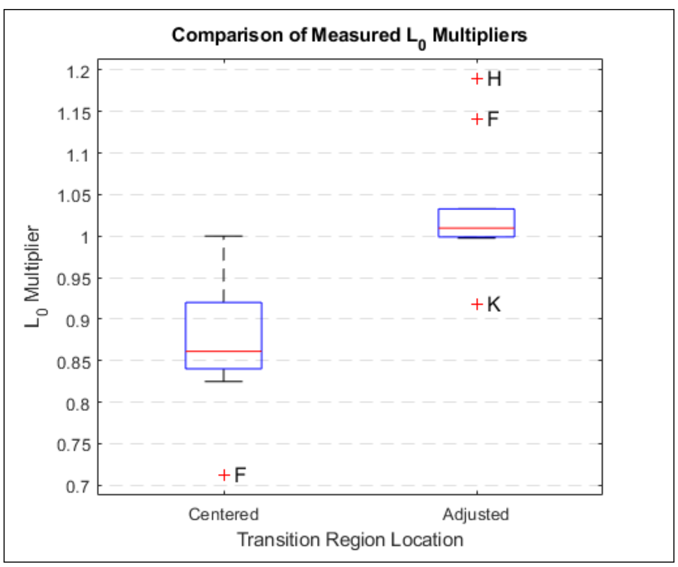


Figure 6.5: Comparison of L_0 Multipliers Measured in Each Stage of Testing

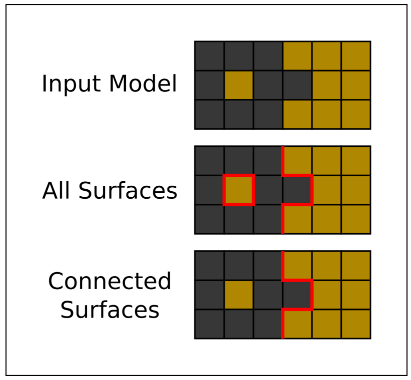


Figure 6.6: Methods for Calculating the Contact Surface Area of a Transition

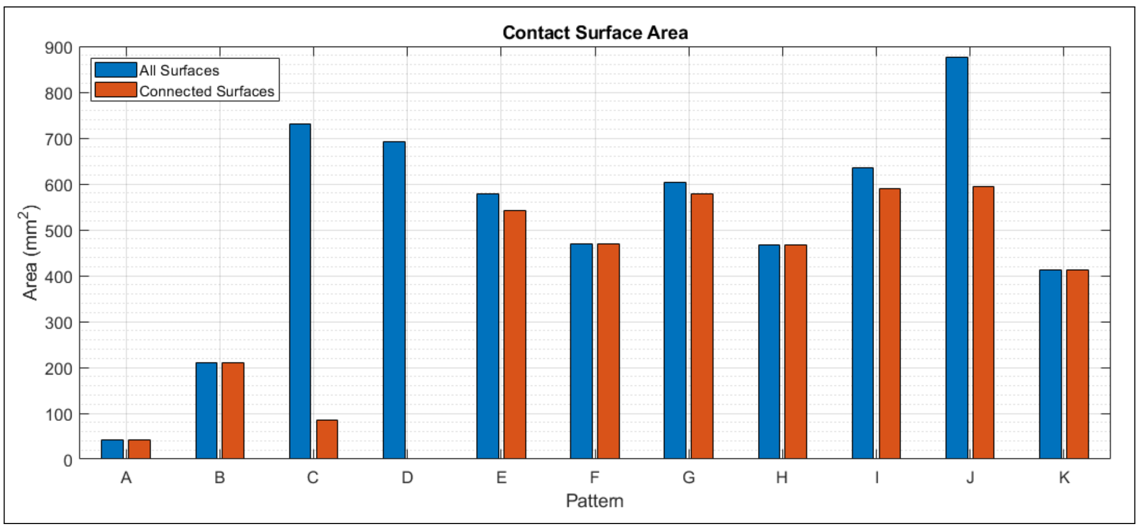


Figure 6.7: Contact Surface Area for Each Transition Type

No strong correlation was observed between the contact surface area and the ultimate tensile strength. The binary (A) and blurred (B) transitions both have a low contact area because they are simply the cross sectional area of the coupon multiplied by the number of material steps. The dithered patterns (C and D) have a high overall surface area due to the large number of disconnected voxels created by the dithering algorithm. However, they have a little to no contact directly between the main rigid and flexible portions of the model because the transition is composed primarily of disconnected voxels. The remaining patterns have variations between their two area values depending on the number of small disconnected components present at the extremes of their transition regions.

Flexible Material Volume Percentage

The volume percentage of flexible material in each model before and after adjusting the transitions was found using VoxelFuse, and the percent change for each was calculated. Figure 6.8 shows these values and compares the changes in each to the changes in ultimate tensile strength.

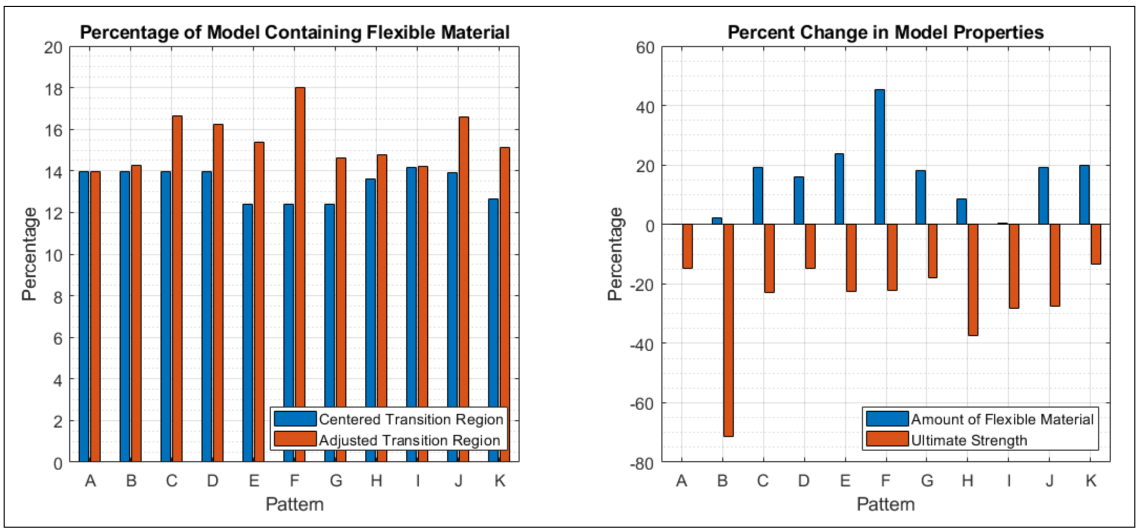


Figure 6.8: Change in the Volume of Flexible Material Present in Each Transition Type

No strong correlation was observed between the changes in volume percentage and ultimate tensile strength.

Discussion

The results from both stages of testing indicate that the addition of a graded material transition has the potential to improve the ultimate tensile strength of a part. The Objet printer used for the tests performed up to this point has strong adhesion between materials. With this printer, the best transition performance was observed in patterns that were closer to a true blur. This indicates that when material adhesion is high, adding a gradual change in part properties is more important than providing a mechanical bond or increasing contact surface area. This is particularly apparent in the case of the dithered patterns, which have almost no direct contact between the rigid and flexible regions of the model. Adding certain types of transitions may even introduce new stress concentrations that weaken the part, as illustrated by the + lattice and large fiber transitions in particular.

In the next set of experiments using the Ultimaker printer, the bonding between materials is expected to be lower. In this case, patterns that provide a better mechanical bond or that increase contact surface area are expected to exhibit increased performance relative to other patterns.

The overall decrease in ultimate tensile strength between the two stages of testing affected the binary interface control samples as well as the samples with modified transitions. As such, the difference appears to be a result of differences in production or testing conditions. Some possibilities during production include use of a different batch of resin, a change in curing time, or use of a different support material removal procedure. Possibilities during testing include differing temperature or humidity levels. Further analysis of these factors is required if absolute stress-strain values for each sample are needed.

The large change in the performance of the blurred transition is not consistent with the results obtained from the other patterns. The blurred pattern also exhibited little to no change in transition location, contact surface area, and flexible material volume percentage between the two stages of testing. Based on these results, it is likely that this pattern was affected by an unidentified factor during production.

Possible factors that could affect this single coupon include its location in the print bed, a malfunction in a print nozzle, or a change in the resin batch having a larger effect than on other coupons due to the higher number of materials present. Further testing of the blurred pattern is planned to investigate the cause of the changes to the blur's performance. This will include producing and testing a set of coupons with blurred transitions that includes multiple transition locations, print bed locations, and print orientations.

The presented approach to modifying the modulus of elasticity to meet a certain goal value was shown to be effective. However, this approach changes the overall percentages of rigid and flexible materials present in the model. This will affect other model properties including mass and moment of inertia. If the structure is a part of an integrated mechanism, properties such as the range of motion or size of gripping surfaces may also be affected. The change in the material composition is also a probable contributor to the changes in ultimate tensile strength. If the production and/or testing factors are identified and mitigated, there may be a stronger correlation between the change in material percentage and ultimate strength. These trade-offs must be considered when selecting how transitions are applied in order to maintain the desired model properties.

Chapter 7

CONCLUSION

This work introduces a series of methods for creating graded material transitions. The first, a Gaussian blur, is representative of a method for creating a true graded transition. The remaining approaches are methods of adapting of a blur to a structure that is compatible with low-cost manufacturing processes which have limits on feature size and/or material count. Test samples and a testing process are introduced to allow the trends in transition design performance to be characterized for a given manufacturing process.

This test results presented in this paper demonstrate that the addition of graded material transitions can improve the tensile strength of a material transition. For a 3D printing process that yields good adhesion between materials, such as the PolyJet process used in this project, it was shown that a graded material design that most closely approximates a blur has a greater impact on improving transition strength than a design which provides a greater mechanical bond or an increase in contact surface area. If supported by the target manufacturing process, directly printing a blurred pattern has the greatest potential to improve part performance. Macro-scale dithered patterns are also a promising option for printing equipment with a limited number of materials. Use of other patterns that provide increased mechanical connection or surface area at the expense of a smooth change in material properties can introduce new stress concentrations which result in additional points of failure, potentially lowering transition performance.

A method is presented for tuning the elastic modulus of a part to the match the original design intent. This method is shown to be effective across different transition designs.

This work also contributes several key features to the VoxelFuse platform to support the generation of the presented graded material transitions, including 3D dithering and periodic structure generation. These expansions and accompanying application examples are made available to allow graded transitions to be easily applied in future work.

Future Work

During testing inconsistencies were observed in the performance of the blurred pattern and between the control samples from the two stages of testing. These are believed to be related to be the result of differences in the manufacturing or testing process. Planned future work includes producing additional samples to better understand the cause of these inconsistencies.

Future work will also include creating test samples using other printing processes. This will allow the changes in transition performance trends for these processes to be identified. Particular focus will be given to low-cost methods in order to support the use of graded material transitions in an educational environment.

The current testing method focuses on the performance of samples under a tensile load perpendicular to the material interface plane. Other potential areas for further testing include changing the angle of the interface plane or testing other loading configurations such as cyclic or compression loading. The failure points of samples could also be assessed through microscope failure analysis.

REFERENCES

Aremu, A. O., J. P. Brennan-Craddock, A. Panesar, I. A. Ashcroft, R. J. Hague, R. D. Wildman and C. Tuck, "A voxel-based method of constructing and skinning conformal and functionally graded lattice structures suitable for additive manufacturing", Additive Manufacturing **13**, 1–13 (2017).

ASTM-D638-14, "Standard Test Method for Tensile Properties of Plastics", ASTM Standards (2014).

Autodesk Inc., "Fusion 360", URL `https://www.autodesk.com/products/fusion-360/overview` (2020).

Bader, "Methods and Apparatus for 3d Printing of Point Cloud Data", (2017).

Brauer, C. and D. Aukes, "Voxel-based CAD framework for planning functionally graded and multi-step rapid fabrication processes", in "Proceedings of the ASME Design Engineering Technical Conference", vol. 2A: 45th D (American Society of Mechanical Engineers, 2019).

Brauer, C., D. Aukes, J. Brauer and C. Jeffries, "VoxelFuse", URL `https://github.com/cdbrauer/VoxelFuse` (2020).

Cham, J. G., S. A. Bailey, J. E. Clark, R. J. Full and M. R. Cutkosky, "Fast and Robust: Hexapedal Robots via Shape Deposition Manufacturing", The International Journal of Robotics Research **21**, 10-11, 869–882 (2002).

Ephtracy, "MagicaVoxel", URL `https://ephtracy.github.io/` (2018).

Hatanaka, M. and M. R. Cutkosky, "Process planning for embedding flexible materials in multi-material prototypes", Proceedings of the ASME Design Engineering Technical Conference **3**, 325–333 (2003).

Hiller, J. and H. Lipson, "Automatic design and manufacture of soft robots", IEEE Transactions on Robotics **28**, 2, 457–466 (2012).

Kaweesa, D. V. and N. A. Meisel, "Material Property Changes in Custom-Designed Digital Composite Structures Due to Voxel Size", Proceedings of the 29th Annual International Solid Freeform Fabrication Symposium pp. 1499–1510 (2018a).

Kaweesa, D. V. and N. A. Meisel, "Quantifying fatigue property changes in material jetted parts due to functionally graded material interface design", Additive Manufacturing **21**, February, 141–149 (2018b).

Kou, X. Y. and S. T. Tan, "Heterogeneous object modeling: A review", CAD Computer Aided Design **39**, 4, 284–301 (2007).

Lou, Q. and P. Stucki, "Fundamentals of 3D halftoning", in "Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)", vol. 1375, pp. 224–239 (Springer Verlag, 1998).

Ma, R. R., L. U. Odhner and A. M. Dollar, "A modular, open-source 3D printed underactuated hand", Proceedings - IEEE International Conference on Robotics and Automation pp. 2737–2743 (2013).

Rossetti, L., L. A. Kuntz, E. Kunold, J. Schock, K. W. Müller, H. Grabmayr, J. Stolberg-Stolberg, F. Pfeiffer, S. A. Sieber, R. Burgkart and A. R. Bausch, "The microstructure and micromechanics of the tendon-bone insertion", Nature Materials **16**, 6, 664–670 (2017).

Stoner, B., J. Bartolai, D. V. Kaweesa, N. A. Meisel and T. W. Simpson, "Achieving Functionally Graded Material Composition Through Bicontinuous Mesostructural Geometry in Material Extrusion Additive Manufacturing", Jom **70**, 3, 413–418 (2018).

Stratasys, "Objet350 and Objet500 Connex3", URL `https://www.stratasys.com/3d-printers/objet-350-500-connex3` (2016).

APPENDIX A

3D DITHERING CODE

```
import PyQt5.QtGui as qg
import sys

import numpy as np

from voxelfuse.materials import material_properties
from voxelfuse.mesh import Mesh
from voxelfuse.plot import Plot
from voxelfuse.primitives import cuboid
from voxelfuse.voxel_model import VoxelModel

from numba import njit

@njit()
def toFullMaterials(voxels, materials, n_materials):
    x_len = voxels.shape[0]
    y_len = voxels.shape[1]
    z_len = voxels.shape[2]

    full_model = np.zeros((x_len, y_len, z_len, n_materials), dtype=np.float32)

    for x in range(x_len):
        for y in range(y_len):
            for z in range(z_len):
                i = voxels[x,y,z]
                full_model[x,y,z,:] = materials[i]

    return full_model

def toIndexedMaterials(voxels, model):
    x_len = model.voxels.shape[0]
    y_len = model.voxels.shape[1]
    z_len = model.voxels.shape[2]

    new_voxels = np.zeros((x_len, y_len, z_len), dtype=np.int32)
    new_materials = np.zeros((1, len(material_properties) + 1), dtype=np.float32)

    for x in range(x_len):
        for y in range(y_len):
            for z in range(z_len):
                m = voxels[x, y, z, :]
                i = np.where(np.equal(new_materials, m).all(1))[0]

                if len(i) > 0:
                    new_voxels[x, y, z] = i[0]
                else:
                    new_materials = np.vstack((new_materials, m))
                    new_voxels[x, y, z] = len(new_materials) - 1

    return VoxelModel(new_voxels, new_materials, model.coords)

@njit()
def addError(model, error, constant, i, x, y, z, x_len, y_len, z_len, error_spread_threshold):
    if y < y_len and x < x_len and z < z_len:
```

```
        high = np.where(model[x, y, z, 1:] > error_spread_threshold)[0]
        if len(high) == 0:
            model[x, y, z, i] += error * constant * model[x, y, z, 0]

@njit()
def ditherOptimized(full_model, use_full, x_error, y_error, z_error, error_spread_threshold):
    x_len = full_model.shape[0]
    y_len = full_model.shape[1]
    z_len = full_model.shape[2]

    for z in range(z_len):
        for y in range(y_len):
            for x in range(x_len):
                voxel = full_model[x, y, z]
                if voxel[0] == 1.0:
                    max_i = voxel[1:].argmax()+1
                    for i in range(1, len(voxel)):
                        if full_model[x, y, z, i] != 0:
                            old = full_model[x, y, z, i]

                            if i == max_i:
                                full_model[x, y, z, i] = 1
                            else:
                                full_model[x, y, z, i] = 0

                            error = old - full_model[x, y, z, i]

                            if use_full:
                                # Based on Fundamentals of 3D Halftoning by Lou and Stucki
                                addError(full_model, error, 4/21, i, x+1, y, z, x_len, y_len, z_len, error_spread_threshold)
                                addError(full_model, error, 1/21, i, x+2, y, z, x_len, y_len, z_len, error_spread_threshold)

                                addError(full_model, error, 4/21, i, x, y+1, z, x_len, y_len, z_len, error_spread_threshold)
                                addError(full_model, error, 1/21, i, x, y+2, z, x_len, y_len, z_len, error_spread_threshold)

                                addError(full_model, error, 1/21, i, x+1, y+1, z, x_len, y_len, z_len, error_spread_threshold)
                                addError(full_model, error, 1/21, i, x-1, y+1, z, x_len, y_len, z_len, error_spread_threshold)

                                addError(full_model, error, 1/21, i, x, y-1, z+1, x_len, y_len, z_len, error_spread_threshold)
                                addError(full_model, error, 1/21, i, x-1, y, z+1, x_len, y_len, z_len, error_spread_threshold)
                                addError(full_model, error, 1/21, i, x, y+1, z+1, x_len, y_len, z_len, error_spread_threshold)
                                addError(full_model, error, 1/21, i, x+1, y, z+1, x_len, y_len, z_len, error_spread_threshold)

```

```
                                addError(full_model, error, 4/21, i, x, y, z+1, x_len, y_len, z_len, error_spread_threshold)
                                addError(full_model, error, 1/21, i, x, y, z+2, x_len, y_len, z_len, error_spread_threshold)
                            else:
                                addError(full_model, error, x_error, i, x+1, y, z, x_len, y_len, z_len, error_spread_threshold)
                                addError(full_model, error, y_error, i, x, y+1, z, x_len, y_len, z_len, error_spread_threshold)
                                addError(full_model, error, z_error, i, x, y, z+1, x_len, y_len, z_len, error_spread_threshold)

    return full_model

def dither(model, radius=1, use_full=True, x_error=0.0, y_error=0.0, z_error=0.0, error_spread_threshold=0.8, blur=True):
    if radius == 0:
        return VoxelModel.copy(model)

    if blur:
        new_model = model.blur(radius)
        new_model = new_model.scaleValues()
    else:
        new_model = model.scaleValues()

    full_model = toFullMaterials(new_model.voxels, new_model.materials, len(material_properties)+1)
    full_model = ditherOptimized(full_model, use_full, x_error, y_error, z_error, error_spread_threshold)

    return toIndexedMaterials(full_model, model)

if __name__ == '__main__':
    app1 = qg.QApplication(sys.argv)

    box_x = 25
    box_y = 40
    box_z = 40

    box1 = cuboid((box_x, box_y, box_z), (0, 0, 0), 1)
    box2 = cuboid((box_x, box_y, box_z), (box_x, 0, 0), 3)
    box3 = cuboid((box_x, box_y, box_z), (box_x*2, 0, 0), 1)
    baseModel = box1 | box2 | box3
    print('Model Created')

    # Process Models
    blurResult = baseModel.blur(int(round(box_x/2)))
    ditherResult = dither(baseModel, int(round(box_x/2)))

    blurMesh = Mesh.fromVoxelModel(blurResult)
    ditherMesh = Mesh.fromVoxelModel(ditherResult)

    plot1 = Plot(blurMesh)
    plot2 = Plot(ditherMesh)

    plot1.show()
    plot2.show()
```

```

    app1.processEvents()
    app1.exec_()
```

APPENDIX B

TRANSITION GENERATION CODE

```python
"""
Copyright 2020
Dan Aukes, Cole Brauer

Generate coupon for tensile testing
"""

import os
import sys
import time
import math
import yaml

import PyQt5.QtGui as qg
from tqdm import tqdm

from voxelfuse.voxel_model import VoxelModel
from voxelfuse.mesh import Mesh
from voxelfuse.plot import Plot
from voxelfuse.primitives import *
from voxelfuse.periodic import *
from voxelfuse.voxel_model import Axes

from dithering.dither import dither

configIDs = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K']

# Set desired outputs
display = False
save = True
export = False

outputFolder = 'stl_files_v4.2_combined'

if __name__=='__main__':
    for configID in configIDs:
        # Load config
        with open("config_files/config_" + configID + ".yaml", 'r') as f:
            try:
                config = yaml.safe_load(f)
            except yaml.YAMLError as exc:
                print(exc)

        # Load settings
        filename = config.get('filename')
        res = config.get('res') # voxels per mm
        couponStandard = config.get('couponStandard') # Start of stl file name
        centerLengthScale = config.get('centerLengthScale') # transition length scale multiplier
        blurRadius = config.get('blurRadius') # mm -- transition region width * 1/2
        materialStep = config.get('materialStep')  # material step size of final result

        blurEnable = config.get('blurEnable', False)
```

```
        ditherEnable = config.get('ditherEnable', False)
        latticeEnable = config.get('latticeEnable', False)
        gyroidEnable = config.get('gyroidEnable', False)

        if ditherEnable:
            ditherType = config.get('ditherType')
            processingRes = config.get('processingRes') # voxels per processed voxel

        if latticeEnable:
            latticeElementFile = config.get('latticeElementFile')
            minRadius = config.get('minRadius')  # 0/1 min radius that results in a printable structure
            maxRadius = config.get('maxRadius')  # 3/5 max radius that results in a viable lattice element

        if gyroidEnable:
            gyroidType = config.get('gyroidType')
            gyroidMaxDilate = config.get('gyroidMaxDilate')
            gyroidMaxErode = config.get('gyroidMaxErode')

        app1 = qg.QApplication(sys.argv)

        # Import coupon components
        print('Importing Files')
        end1 = VoxelModel.fromMeshFile('coupon_templates/' + couponStandard + '-End.stl', (0, 0, 0), resolution=res).fitWorkspace()
        center = VoxelModel.fromMeshFile('coupon_templates/' + couponStandard + '-Center-2.stl', (0, 0, 0), resolution=res).fitWorkspace()
        end2 = end1.rotate90(2, axis=Axes.Z)
        center.coords = (end1.voxels.shape[0], round((end1.voxels.shape[1] - center.voxels.shape[1]) / 2), 0)
        end2.coords = (end1.voxels.shape[0] + center.voxels.shape[0], 0, 0)

        # Trim center
        center_cross_section = VoxelModel(center.voxels[0:2, :, :], 3).fitWorkspace()
        centerLength = center.voxels.shape[0]
        centerWidth = center_cross_section.voxels.shape[1]
        centerHeight = center_cross_section.voxels.shape[2]

        centerCoordsOffset = (
        center.coords[0], center.coords[1] + round(((center.voxels.shape[1] - centerWidth) / 2)), center.coords[2])
        center = cuboid((centerLength, centerWidth, centerHeight), centerCoordsOffset)

        # Set materials
        end1 = end1.setMaterial(1)
        end2 = end2.setMaterial(1)
        center = center.setMaterial(2)

        # Combine components
        coupon = end1 | center | end2
```

```
        coupon = coupon.setMaterial(1)

        # Scaled center
        newCenterLength = round(centerLength * centerLengthScale)
        centerCoordsOffset = (center.coords[0] + round((centerLength - newCenterLength) / 2), center.coords[1], center.coords[2])
        center = cuboid((newCenterLength, centerWidth, centerHeight), centerCoordsOffset)
        center = center.setMaterial(2)
        coupon = center | coupon

        coupon_input = VoxelModel.copy(coupon)

        start = time.time()

        # Generate transition regions
        transition_1 = cuboid((blurRadius * res * 2, coupon.voxels.shape[1], coupon.voxels.shape[2]), (center.coords[0] - (blurRadius * res), 0, 0), 3)
        transition_2 = cuboid((blurRadius * res * 2, coupon.voxels.shape[1], coupon.voxels.shape[2]), (center.coords[0] - (blurRadius * res) + newCenterLength, 0, 0), 3)
        transition_regions = transition_1 | transition_2
        transition_regions = transition_regions.getComponents()

        # Generate lattice elements
        latticeSize = None
        lattice_elements = None
        if latticeEnable:
            # Import Models
            lattice_model = VoxelModel.fromVoxFile("lattice_elements/" + latticeElementFile + '.vox')
            latticeSize = lattice_model.voxels.shape[0]
            print('Lattice Element Imported')

            # Generate Dilated Lattice Elements
            lattice_elements = [VoxelModel.emptyLike(lattice_model)]
            for r in range(minRadius, maxRadius + 1):
                lattice_elements.append(lattice_model.dilate(r))
            lattice_elements.append(cuboid(lattice_model.voxels.shape))
            print('Lattice Elements Generated')

        elif gyroidEnable:
            # Import Models
            s = center.voxels.shape[2]

            if gyroidType == 2:
                lattice_model_1, lattice_model_2 = schwarzP((s,s,s), s)
            elif gyroidType == 3:
                lattice_model_1, lattice_model_2 = schwarzD((s,s,s), s)
            elif gyroidType == 4:
                lattice_model_1, lattice_model_2 = FRD((s,s,s), s)
            else: # gyroidType == 1
                lattice_model_1, lattice_model_2 = gyroid((s,s,s), s
```

```
)

            latticeSize = s
            print('Lattice Element Imported')

            # Generate Dilated Lattice Elements
            lattice_elements = [VoxelModel.emptyLike(lattice_model_1)]
            for r in range(0, gyroidMaxErode):
                lattice_elements.append(lattice_model_1.difference(lattice_model_2.dilate(gyroidMaxErode - r)))
            for r in range(0, gyroidMaxDilate + 1):
                lattice_elements.append(lattice_model_1.dilate(r))
            lattice_elements.append(cuboid(lattice_model_1.voxels.shape))
            print('Lattice Elements Generated')

        # Generate transitions
        for c in range(transition_regions.numComponents):
            print('Component #' + str(c+1))
            transition = coupon_input & (transition_regions.isolateComponent(c+1))
            transitionCenter = transition.getCenter()
            transition = transition.fitWorkspace()
            print(transitionCenter)

            if blurEnable: # Blur materials
                print('Blurring')
                transition_scaled = transition.blur(blurRadius*res)   # Apply blur
                transition_scaled = transition_scaled.scaleValues()   # Cleanup values
                transition_scaled = transition_scaled.setCenter(transitionCenter)   # Center processed model on target region
                transition = transition_scaled & transition   # Trim excess voxels

            elif ditherEnable: # Dither materials
                print('Dithering')
                x_len = int(transition.voxels.shape[0])
                y_len = int(transition.voxels.shape[1])
                z_len = int(transition.voxels.shape[2])

                transition_scaled = transition.blur(blurRadius*res*1.5)
                transition_scaled = transition_scaled.scale((1 / processingRes))   # Reduce to processing scale and dilate to compensate for rounding errors

                if ditherType == 2:
                    transition_scaled = dither(transition_scaled, blurRadius*(res/processingRes), blur=False, use_full=False, y_error=0.8, x_error=0.8)   # Apply Dither
                elif ditherType == 3:
                    transition_scaled = dither(transition_scaled, blurRadius*(res/processingRes), blur=False, use_full=False, y_error=0.8)   # Apply Dither
```

```python
                else: # ditherType == 1
                    transition_scaled = dither(transition_scaled, blurRadius * (res / processingRes), blur=False)  # Apply Dither

                transition_scaled = transition_scaled.scaleValues()                            # Cleanup values
                transition_scaled = transition_scaled.scaleToSize(x_len, y_len, z_len)                            # Increase to original scale
                transition_scaled = transition_scaled.setCenter(transitionCenter)                            # Center processed model on target region
                transition = transition_scaled & transition                            # Trim excess voxels

            elif latticeEnable or gyroidEnable:
                print('Lattice')

                boxX = math.ceil(transition.voxels.shape[0] / latticeSize)
                boxY = math.ceil(transition.voxels.shape[1] / latticeSize)
                boxZ = math.ceil(transition.voxels.shape[2] / latticeSize)
                print([boxX, boxY, boxZ])

                lattice_locations = transition.scaleToSize(boxX, boxY, boxZ)
                lattice_locations = lattice_locations.blur(blurRadius*(res/latticeSize))
                lattice_locations = lattice_locations.scaleValues()
                lattice_locations = lattice_locations - lattice_locations.setMaterial(2)
                lattice_locations = lattice_locations.scaleNull()

                # Convert processed model to lattice
                lattice_result = VoxelModel.emptyLike(lattice_locations)

                for x in tqdm(range(boxX), desc='Adding lattice elements'):
                    for y in range(boxY):
                        for z in range(boxZ):
                            i = lattice_locations.voxels[x, y, z]
                            density = lattice_locations.materials[i, 0] * (1 - lattice_locations.materials[i, 1])

                            if density < 1e-10:
                                r = 0
                            elif density > (1 - 1e-10):
                                r = len(lattice_elements) - 1
                            else:
                                r = round(density * (len(lattice_elements) - 3)) + 1

                            r = int(r)

```

```
                        locationOffset = round((lattice_elements[r].voxels.shape[0] - latticeSize) / 2)

                        x2 = (x * latticeSize) - locationOffset
                        y2 = (y * latticeSize) - locationOffset
                        z2 = (z * latticeSize) - locationOffset

                        lattice_elements[r].coords = (x2, y2, z2) # Do not use setCoords here

                        lattice_result = lattice_result.union(lattice_elements[r])

            lattice_result = lattice_result.setMaterial(1)
            lattice_result = lattice_result.setCenter(transitionCenter)   # Center processed model on target region
            transition_scaled = lattice_result & transition          # Trim excess voxels
            transition = transition_scaled | transition.setMaterial(2)

        transition = transition & coupon    # Trim excess voxels
        coupon = transition | coupon        # Add to result

    coupon = coupon.round(materialStep)
    coupon = coupon.removeDuplicateMaterials()
    coupon.resolution = res

    end = time.time()
    processingTime = (end - start)
    print("Processing time = %s" % processingTime)

    if display:
        # Create mesh data
        print('Meshing')
        mesh1 = Mesh.fromVoxelModel(coupon_input.difference(transition_regions).setMaterial(3) | coupon, resolution=res)

        # Create plot
        print('Plotting')
        plot1 = Plot(mesh1, grids=True, drawEdges=True, positionOffset = (35, 2, 0), viewAngle=(50, 40, 200), resolution=(720, 720), name=filename)
        plot1.show()
        app1.processEvents()

    if save:
        try:
            os.mkdir(outputFolder)
        except OSError:
            print('Output folder already exists')
        else:
            print('Output folder successfully created')

        print('Saving')
        coupon.saveVF(outputFolder + '/output_' + filename)

```

```
        if export:
            print('Exporting')

            try:
                os.mkdir(outputFolder + '/stl_output_' + filename)
            except OSError:
                print('Output folder already exists')
            else:
                print('Output folder successfully created')

            for m in range(1, len(coupon.materials)):
                current_mesh = Mesh.fromVoxelModel(coupon.
    isolateMaterial(m), resolution=res)
                current_mesh.export((outputFolder + '/stl_output_' +
     filename +'/' + couponStandard + '_mat_' + str(m) + '_' + str(
    coupon.materials[m, 2]) + '.stl'))

    print('Finished')
    app1.exec_()
```

APPENDIX C

CONTACT SURFACE MEASUREMENT CODE

```
"""
Copyright 2020
Dan Aukes, Cole Brauer

1. Isolate a region of a model
2. Select one material
3. Find the number of faces where this material meets another material
4. Convert number of faces to surface area
"""

# Import Libraries
import PyQt5.QtGui as qg
import sys
import numpy as np
from tqdm import tqdm

from voxelfuse.voxel_model import VoxelModel
from voxelfuse.primitives import *
from voxelfuse.mesh import Mesh
from voxelfuse.plot import Plot

# Start Application
if __name__=='__main__':
    app1 = qg.QApplication(sys.argv)

    transitionWidth = 12
    transitionCenter = 71.1
    materialToMeasure = 1

    largestOnly = True # Only look at the largest component of each material
    cleanup = False # Remove duplicate materials
    display = False # Display output

    # Open File
    file = 'stl_files_v4.2_combined/output_K'
    model = VoxelModel.openVF(file)
    model.resolution = 5 # Manually set resolution if not set in file
    res = model.resolution

    # Define a region in which to calculate the contact area
    testRegionSize = (round(transitionWidth*res*1.75), model.voxels.shape[1], model.voxels.shape[2])
    testRegionLocation = (model.coords[0] + round(transitionCenter*res) - round(testRegionSize[0]*.5), model.coords[1], model.coords[2])
    test_region = cuboid(testRegionSize, testRegionLocation, material=3, resolution=res)

    # Cleanup operations
    if cleanup:
        model.materials = np.round(model.materials, 3)
        model = model.removeDuplicateMaterials()

    # Crop the model to the region of interest
```

```
    model_cropped = VoxelModel.emptyLike(model)
    if largestOnly:
        # Isolate the region of the model that intersects with the test region
        model_cropped_all_components = model & test_region

        # Loop through each material in model
        for m in range(1, len(model_cropped.materials)):
            model_current_material = model_cropped_all_components.isolateMaterial(m)
            model_current_material = model_current_material.getComponents()

            # Find the volume of each component made of this material
            componentVolumes = np.zeros(model_current_material.numComponents+1)
            for c in range(1, model_current_material.numComponents+1):
                componentVolumes[c], _ = model_current_material.getVolume(component=c)

            # Identify the largest component and add to cropped model
            largestComponent = componentVolumes.argmax()
            model_cropped = model_current_material.isolateComponent(largestComponent) | model_cropped
    else:
        # Isolate the region of the model that intersects with the test region
        model_cropped = model & test_region

    # Isolate the material of interest
    model_single_material = model_cropped.isolateMaterial(materialToMeasure)

    # Find exterior voxels
    surfaceVoxelsArray = model_single_material.difference(model_single_material.erode(radius=1, connectivity=1)).voxels

    x_len = surfaceVoxelsArray.shape[0]
    y_len = surfaceVoxelsArray.shape[1]
    z_len = surfaceVoxelsArray.shape[2]

    # Create list of exterior voxel coordinates
    surfaceVoxelCoords = []
    for x in tqdm(range(x_len), desc='Finding exterior voxels'):
        for y in range(y_len):
            for z in range(z_len):
                if surfaceVoxelsArray[x, y, z] != 0:
                    surfaceVoxelCoords.append([x, y, z])

    # Find number of contact surfaces for each surface voxel
    totalSurfaceCount = 0
    for voxel_coords in tqdm(surfaceVoxelCoords, desc='Checking for contact surfaces'):
        x = voxel_coords[0]
```

```
        y = voxel_coords[1]
        z = voxel_coords[2]

        if x+1 < x_len:
            if (model_cropped.voxels[x+1, y, z] != 0) and (model_cropped.voxels[x+1, y, z] != materialToMeasure):
                totalSurfaceCount = totalSurfaceCount + 1
        if x-1 >= 0:
            if (model_cropped.voxels[x-1, y, z] != 0) and (model_cropped.voxels[x-1, y, z] != materialToMeasure):
                totalSurfaceCount = totalSurfaceCount + 1

        if y+1 < y_len:
            if (model_cropped.voxels[x, y+1, z] != 0) and (model_cropped.voxels[x, y+1, z] != materialToMeasure):
                totalSurfaceCount = totalSurfaceCount + 1
        if y-1 >= 0:
            if (model_cropped.voxels[x, y-1, z] != 0) and (model_cropped.voxels[x, y-1, z] != materialToMeasure):
                totalSurfaceCount = totalSurfaceCount + 1

        if z+1 < z_len:
            if (model_cropped.voxels[x, y, z+1] != 0) and (model_cropped.voxels[x, y, z+1] != materialToMeasure):
                totalSurfaceCount = totalSurfaceCount + 1
        if z-1 >= 0:
            if (model_cropped.voxels[x, y, z-1] != 0) and (model_cropped.voxels[x, y, z-1] != materialToMeasure):
                totalSurfaceCount = totalSurfaceCount + 1

    print('\nNumber of contact surfaces: ' + str(totalSurfaceCount))
    print('Surface area: ' + str(totalSurfaceCount*(1/res)*(1/res)) + ' mm^2')

    if display:
        mesh = Mesh.fromVoxelModel(model_single_material.setMaterial(4) | model_cropped.setMaterial(3) | model)

        # Create Plot
        plot1 = Plot(mesh, grids=True)
        plot1.show()
        app1.processEvents()
        app1.exec_()
```