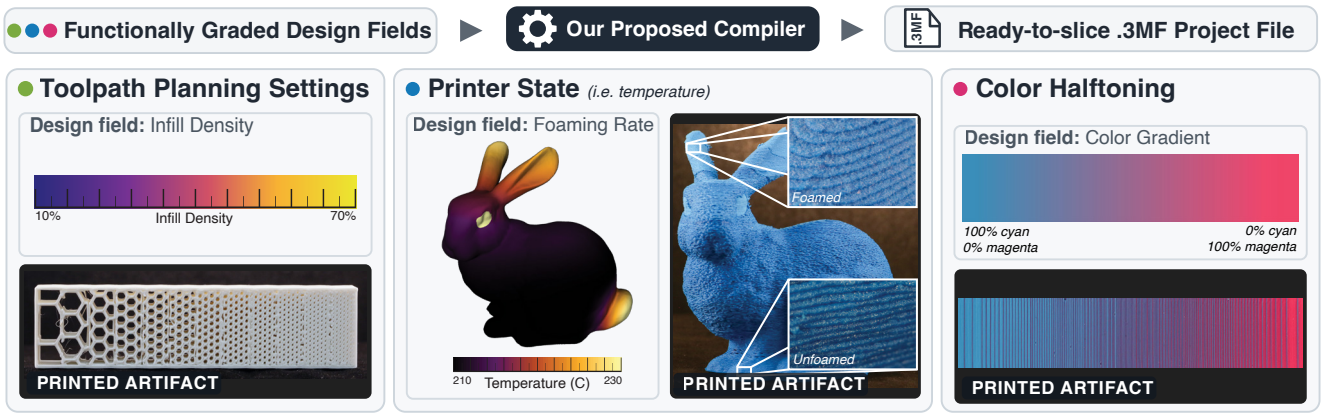
Functionally Grading the Slicing Process by Compiling Design Intent into Slicer Projects

Charles Wade
charles.wade@colorado.edu
University of Colorado Boulder
Boulder, Colorado, USA
Draper Scholars, The Charles Stark Draper Laboratory, Inc
Cambridge, Massachusetts, USA

Devon Beck
dbeck@draper.com
The Charles Stark Draper Laboratory, Inc
Cambridge, Massachusetts, USA

Robert MacCurdy*
maccurdy@colorado.edu
University of Colorado Boulder
Boulder, Colorado, USA



Abstract

Functional gradients give designers finer control over part behavior by varying structure, material, or process conditions across an object. Yet functionally graded fabrication is often treated as a problem of grading geometry or material distribution, rather than grading the slicing and fabrication process itself. In material-extrusion printing, many functional effects arise directly from slicer-controlled mechanisms: local toolpath planning, surface treatment, material assignment, color mixing, and printer state can all vary across space. Mainstream FFF slicers already expose these mechanisms as settings, but they require users to reconstruct heterogeneous intent as manually assigned mesh regions in their GUI. This paper presents slicer project compilation: a fully automated workflow that lowers heterogeneous implicit designs into slicer-native `.3mf` projects with embedded sub-meshes, settings, recipes, and process-state assignments. The compiler partitions spatial attributes into finite regions, extracts aligned sub-meshes, and serializes those regions into the project dialect required by the target slicer while preserving native toolpath planning, preview, support generation, and printer-profile infrastructure. Our method connects heterogeneous design representations to fully featured slicers, enabling automated and scalable fabrication workflows that retain existing slicer ecosystems. We demonstrate this approach across three classes of slicing and fabrication parameters: settings meshes, virtual extrusion, and color or material halftoning. We also introduce calibrated translation models for temperature responsive foaming TPU and PLA, allowing high-level density and Shore-hardness fields to drive fabrication-ready process fields. Across printed examples, the compiler generates ready-to-slice projects for graded toolpath settings, foaming-filament properties, combined texture and process-state control, and color or material-mixture halftoning, replacing more than 2,500 repetitive manual slicer interactions. Our open-source implementation provides a reusable foundation for functionally graded FFF research and applications built on existing slicer ecosystems.

CCS Concepts

• **Computing methodologies** → **Volumetric models**; Mesh geometry models; • **Applied computing** → **Computer-aided manufacturing**; **Computer-aided design**.

Keywords

computer-aided design, additive manufacturing, volumetric design, functionally graded materials, slicing

1 Introduction

Heterogeneous fabrication aims to make spatial variation a first-class design variable. A printed object may vary stiffness, density, color, surface texture, or infill structure across its volume, and these variations can produce functional behavior that uniform material assignment cannot. Modern additive manufacturing workflows provide many of the low-level mechanisms needed to realize such

*Corresponding Author

variation: slicers can assign different settings to different model regions, multi-tool systems can assign different materials, color-mixing workflows can approximate continuous color, and G-code can control printer state during fabrication. These fabrication mechanisms provide the necessary building blocks, but current slicing workflows expose them through interfaces that make dense heterogeneous control impractical: users must manually decompose the design, assign settings region by region, or rely on brittle G-code post-processing. This creates a gap between heterogeneous design methods, which can express continuous spatial intent, and mature slicing tools, which can fabricate local variation only after that intent has been manually entered into the slicer.

Conventional slicers were built around discrete meshes and user-authored settings, not continuous heterogeneous designs. A designer who specifies a volumetric gradient must usually translate that intent into slicer-readable form by hand. In practice, this means partitioning geometry into regions, exporting and aligning many sub-meshes, assigning per-region settings or materials in the slicer interface, and encoding process-state changes through printer-specific scripts or post-processing. This manual lowering ties the design to a particular fabrication setup and limits scalability as the number of regions, attributes, or target slicers increases. Mature slicers provide valuable path planning, preview, support generation, and printer-profile infrastructure, but their interfaces do not directly consume field-based heterogeneous designs.

Heterogeneous modeling is a well-studied upstream design problem, and state-of-the-art field-based representations can encode geometry and spatial attributes as functions over the object domain. Such representations allow a design to express high-level intent, such as target density, Shore hardness, color, infill density, or surface texture, without discretizing that intent into slicer objects. However, the field must still be lowered into the concrete project structures that the target slicer can consume: sub-meshes, setting overrides, material assignments, virtual recipes, logical tools, and printer-specific commands.

To address this gap, we introduce slicer project compilation as an automated lowering step from heterogeneous attribute fields to slicer-native project files. The compiler takes an attribute-modeled implicit design as input, partitions the required spatial fields into finite regions, extracts those regions as aligned sub-meshes, and writes a configured `.3mf` project with embedded settings, material assignments, recipe definitions, and virtual-tool mappings. The downstream slicer remains responsible for toolpath generation. Thus, the proposed workflow does not replace mature slicers; it leverages them as compilation targets.

The same project-generation framework can target several forms of heterogeneous fabrication control. In this paper, we demonstrate the approach with three representative slicer mechanisms. First, the compiler converts toolpath-planning fields, such as infill density or fuzzy-skin parameters, into region-specific slicer setting overrides. Second, it converts process-state fields into logical tool assignments and custom toolchange commands, allowing slicer-generated toolchange events to control G-code printer states such as nozzle temperature. Third, it converts material-fraction or sRGB color fields into mixed-filament recipe assignments for existing color-mixing workflows. These targets differ in the project data they serialize, but they share the same core abstraction: continuous spatial attributes are reduced to labeled project regions that a conventional slicer can load, preview, and slice.

We evaluate our proposed method with a suite of printed examples that exercise heterogeneous control across slicer-planning settings, process-state assignments, and color or material-mixture workflows. These examples test whether the compiler can preserve spatial fields as slicer-readable project assignments rather than requiring users to recreate those assignments manually in the slicer interface. We also evaluate calibrated translation models used before compilation to resolve high-level property fields into the process controls required for fabrication, allowing object intent to drive machine-state assignments automatically. Across the results, settings-mesh examples demonstrate local control over slicer-generated toolpaths, virtual-extrusion examples demonstrate property-driven process-state control, and halftoning project examples demonstrate compilation of material-fraction and color fields to existing slicer workflows. Together, these results show that slicer project compilation can bridge field-based heterogeneous design and mature slicer workflows without requiring a custom-built slicer for each fabrication task.

2 Related Work

Prior literature reviews show that heterogeneous design, including volumetric variation and functional grading, can improve printed artifacts and expand the design space of additive manufacturing [Li et al. 2020; Loh et al. 2018]. A functionally graded object may vary material composition, color, density, stiffness, surface texture, toolpath structure, or process state across space. This breadth creates a translation problem: heterogeneous intent can originate in field-based design representations, but fabrication may require local composition control, slicer setting overrides, printer-state control, or color halftoning. We therefore review prior work in four areas that define this interface: field-based representations and compilation, local composition control, spatial control of slicer settings and process state, and color halftoning for material extrusion. Across these areas, we focus on the gap our work addresses: using mature slicers and readily available desktop FFF printers to realize heterogeneous designs without custom slicers, manually assigned regions, or post-processed G-code.

2.1 Field-Based Design Representations and Compilation

Field-based design representations support heterogeneous design by specifying spatial information that boundary meshes represent poorly. A mesh can describe an object's exterior surface, and multiple meshes can describe discrete regions, but a smooth volumetric gradient must be approximated by many separate boundaries before a conventional slicer can assign materials or settings. Field-based representations avoid this reconstruction step by describing geometry, composition, color, or other attributes as functions over space. Programmable fabrication systems such as OpenFab use this idea to express material variation procedurally [Vidimče et al. 2013], while OpenVCAD represents implicit geometry and volumetric material composition as spatial fields for multi-material

fabrication [Wade et al. 2025a, 2024]. These representations let designers describe heterogeneous intent directly, but fabrication still requires a compiler that converts fields into the concrete inputs expected by a target manufacturing workflow.

Recent heterogeneous attribute modeling extends this field-based view by treating heterogeneous fabrication as a multi-stage compilation problem [Wade et al. 2026]. Rather than representing only geometry or material composition, the approach describes objects using named, typed spatial attributes that may encode high-level design intent, such as target mechanical response, hardness, or color. Translation models then map these intent-level attributes into process-specific realization fields, such as material recipes. This framing is important for our work because it separates what the designer specifies from the machine-specific structures that fabrication requires. The remaining compilation step must convert the resolved realization fields into the concrete project data consumed by the target workflow.

2.2 Local Composition Control for Functionally Graded Fabrication

Local composition control provides the most direct route to functionally graded fabrication. In these systems, the printer changes the relative amounts of two or more base materials across an object, producing spatial variation in appearance or material behavior. Voxel-based material jetting demonstrates this idea at high spatial resolution by assigning material channels throughout a volume, and programmable fabrication pipelines have used this capability to express complex multi-material distributions [Vidimče et al. 2013; Wade et al. 2024]. These systems establish the value of volumetric material control but focus only on material-jetting workflows and are not directly applicable to FFF slicing control.

Material-extrusion systems approach local composition control through a different mechanism. Multi-feed and mixing hotend systems combine filaments during deposition, allowing the commanded mixture ratio to vary during a print. Kennedy and Christ demonstrate that active in-situ mixing can blend polymer filaments during fused filament fabrication [Kennedy and Christ 2020]. Similarly, Green et al. use an active-mixing hotend to spatially control mixture ratios and fabricate graded compositions [Green et al. 2023]. These works establish mixing extrusion as a viable route for graded composition in FFF. They also motivate the fabrication-planning problem that follows: a spatial composition field must still be converted into printer instructions that coordinate material ratios with the toolpath. Garland and Fadel demonstrate this challenge on an off-the-shelf mixing extrusion system by discretizing a gradient into regions, assigning those regions as virtual extruders in Slic3r, and post-processing the resulting G-code to replace toolchange commands with mixture-ratio commands [Garland and Fadel 2015]. Leoni et al. further frame this problem as a gap between functionally graded design and material-extrusion fabrication [Leoni et al. 2023]. These works highlight that local composition control in FFF depends not only on extrusion hardware, but also on representations and software workflows that can translate graded design intent into slicer-compatible assignments.

Direct-ink-write systems provide another important class of local composition control. Active mixing has enabled graded fluids, reactive materials, ceramics, elastomers, and dielectric composites [Ober et al. 2015; Ortega et al. 2019; Pelz et al. 2021]. Duncan et al., for example, use active mixing of nanocomposite inks to fabricate low-loss graded dielectrics with spatially tailored electromagnetic properties [Duncan et al. 2023]. These examples show that continuous material variation can produce application-specific functional behavior. However, they also show that DIW composition grading is typically developed as an application-specific fabrication workflow, where the material formulation, printhead behavior, path design, and process commands must be coordinated for each system rather than specified through a reusable heterogeneous design.

These systems establish local composition control as a useful fabrication primitive. The remaining challenge is how to translate a spatial composition field into slicer-compatible regions and material assignments without requiring the designer to reconstruct the gradient as a collection of manually assigned meshes when moving between tools.

2.3 Spatial Control of Slicer Settings and Process State

In addition to composition variation, slicers also shape local behavior through the toolpaths they generate: infill density, perimeter count, surface texture, and related settings can change stiffness, strength, weight, and appearance while using a single material. Prior work shows that toolpath structure and infill design strongly affect mechanical performance, and several methods use stress, topology, or other fields to guide path orientation and material placement [Liu et al. 2024; Sales et al. 2021; Xia et al. 2020]. These methods make toolpath planning itself part of the design space. Modern slicers expose a related capability through object-level settings, modifier meshes, and painted regions, which allow users to apply different path-generation rules to different parts of a build [Prusa Research ndb]. However, these interfaces remain largely interactive. A designer can assign different infill densities to different imported meshes, but a high-resolution spatial field must first be reconstructed as many aligned models and then assigned settings region by region.

Surface texture provides a compact example of the same issue. Fuzzy skin perturbs exterior toolpaths to produce a roughened surface, and slicers expose this behavior as a global setting or as a locally painted surface effect [Prusa Research nda; Vesco and Salvi 2025]. Thus, surface texture is a slicer-level property rather than a material-composition property. Yet the slicer interface does not provide a general field representation for smoothly varying texture parameters over a model. As with infill, the available slicing feature is useful for functional grading, but the interface between spatial design intent and path-generation parameters remains indirect.

Process-state control broadens this pattern beyond slicer settings. A printer can vary local properties by changing machine commands during fabrication, including flow rate, print speed, cooling, laser power, energy input, or nozzle temperature. Metal additive manufacturing has used site-specific process control to vary melt-pool behavior, geometry, porosity, and material response [Borish et al. 2022; Gibson et al. 2019; Yuan et al. 2022]. In material extrusion, foaming filaments provide a single-material route to graded

density because extrusion temperature and flow compensation affect expansion, porosity, and mechanical response [Damanpack et al. 2021; Ozdemir and Doubrovski 2023; Tammaro et al. 2022]. These examples show that functional grading can come from printer state, in addition to material mixtures and toolpath settings.

However, process-state fields introduce a planning problem that ordinary slicer controls rarely address. Some commands can change instantaneously, while others require stabilization time, material flow, or thermal response before the printed material reaches the intended state. Gradient-informed slicing addresses this problem by making the spatial field part of toolpath planning, including for mixture ratios, tool changes, and temperature responsive foaming materials [Wade et al. 2025b]. This approach shows the value of custom gradient-aware slicers, but it also illustrates a tradeoff: a custom slicer must reimplement many surrounding capabilities that mature slicers already provide, including support generation, skin and infill strategies, printer profiles, modifier semantics, and stable project workflows.

Our proposed method follows the complementary direction of working within full-featured slicer workflows rather than replacing them. Existing slicers already contain abstractions for local control, including object-level settings, modifier regions, tool assignments, and tool-change events. Prior gradient-informed workflows also show that tool-like channels can represent fabrication states rather than only physical extruders [Wade et al. 2025b]. The open problem is how to connect spatial design fields to these slicer abstractions automatically, without reducing the workflow to manual region assignment or requiring a bespoke slicer for each graded fabrication task.

2.4 Color Halftoning for Material Extrusion

Color provides another instance of spatially varying fabrication intent, but it differs from the process-state and slicer-setting controls because the color reproduction problem has a long computational history. Full-color 3D printing must map continuous colors to a finite set of printable materials or deposition states. Prior work has addressed this mapping through color management, contoning, surface halftoning, voxel assignment, and line-based halftoning [Babaei et al. 2017; Yuan et al. 2021]. In material extrusion, Reiner et al. approximate continuous tones with dual-color deposition patterns [Reiner et al. 2014], while Kuipers et al. adapt hatching and line-based halftoning to the path structure of fused deposition modeling [Kuipers et al. 2017, 2018]. Song et al. further show colored fused filament fabrication using layered color strata and mixing-nozzle hardware [Song et al. 2019]. These methods show that discrete filaments or material states can approximate continuous color, but they focus primarily on the color assignment, optical model, or deposition strategy.

Recent slicers make filament-based color workflows more accessible by exposing color-mixing and halftoning features directly in open-source slicing ecosystems. Prusa ColorMix and Orca FullSpectrum provide practical mechanisms for assigning printable color mixtures within slicer workflows [Prusa Research 2026; ratdoux 2026]. These systems are important because they turn color halftoning from a standalone research pipeline into a slicer-level fabrication capability. However, the remaining interface problem mirrors the challenges with local settings and process state: a full-color design must still be converted into slicer-level assignments that the downstream halftoning method can consume.

This places color within the same interface gap as other heterogeneous fabrication parameters. Prior work provides effective color-assignment and halftoning methods, and recent slicers provide practical mechanisms for fabricating those assignments. The missing step is a general way to connect full-color heterogeneous designs to slicer-level color workflows without requiring the designer to manually reconstruct the design as printable regions or recipes.

2.5 Summary

The challenge with heterogeneous fabrication workflows is neither field representations nor slicer features in isolation. Prior work provides rich design mechanisms for local composition control, process-state control, color reproduction, and heterogeneous modeling. Existing slicers also provide mature path-generation features and local control mechanisms. The remaining challenge is an automated lowering step that connects field-based heterogeneous intent to a format a mature slicing program can ingest. This work addresses that gap by generating slicer-ready projects in which the sub-meshes, settings, virtual tool assignments, and color assignments are embedded before slicing.

3 Methods

3.1 Overview: Slicer Project Compilation

We define slicer project compilation as the conversion of an attribute-modeled implicit design into a slicer-native `.3mf` project whose sub-regions carry the settings, process-state assignments, or material-mixture recipes required by the target workflow. The slicer project compiler does not replace the downstream slicer. Instead, it outputs a project file that the slicer can load, preview, slice, and convert to G-code using its native toolpath generator.

Our proposed compiler takes as input an implicit heterogeneous object whose compiler-required attributes have been authored directly or resolved before compilation. Required attributes are the fields that a selected compilation target needs in order to generate slicer-ready project data, and they vary by compilation type. For example, one compiler may require slicer-planning settings such as infill density, another may require process-state setpoints such as nozzle temperature or flow-rate compensation, and another may require color or material-fraction fields. The compiler first partitions one or more attributes into a finite set of spatial regions. It then extracts each nonempty region as a sub-mesh and labels that sub-mesh with representative attribute values. The final serialization step is target-specific: the compiler writes the metadata, tool assignments, recipe definitions, and project settings required by the selected slicer-project dialect.

Our compiler abstraction is agnostic to the selected compilation target. As shown in Figure 1, we demonstrate this abstraction with three compiler targets, while preserving a shared partitioning and sub-mesh generation stage. Settings meshes assign slicer-planning

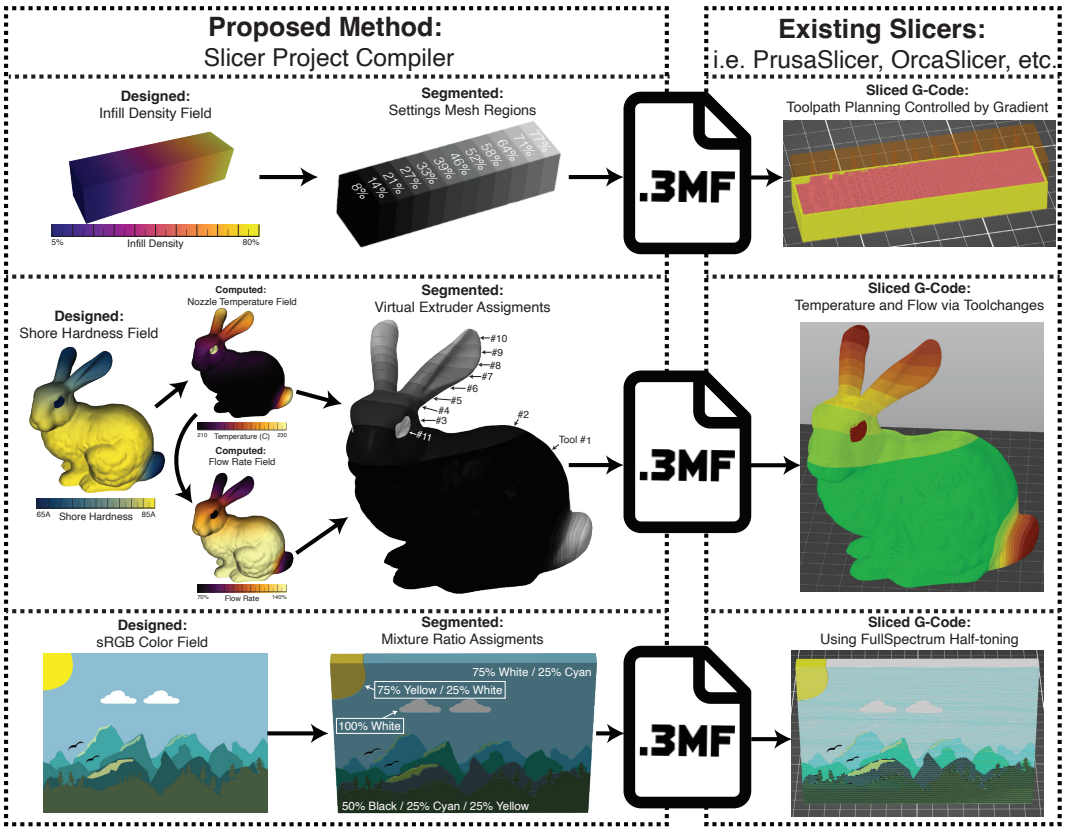


Figure 1: Slicer project compilation lowers heterogeneous attribute fields into slicer-native `.3mf` projects. The compiler partitions resolved fields into labeled sub-meshes and serializes them as settings-mesh regions, virtual-extruder assignments, or mixture-ratio assignments. Existing slicers then perform their native toolpath planning, toolchange emission, or halftoning workflows from the generated project files.

parameters, such as infill density or fuzzy-skin settings, to generated regions before toolpath planning. Virtual extrusion assigns regions to logical tools and uses custom toolchange templates to redirect slicer toolchange events into process-state commands. Color and material halftoning projects assign regions to virtual mixed-filament recipes that ColorMix or FullSpectrum-style workflows convert into filament-level halftoning. These targets differ in the assignments they write, but they share the same field partitioning and sub-mesh generation stage.

The targets can also be combined when the slicer-project dialect supports the required assignments. For example, a foaming-filament workflow may use virtual extrusion to control temperature and flow rate, while also using settings meshes to apply spatially varying fuzzy skin. In such cases, the compiler partitions the participating attributes jointly so that each generated region carries an identical assignment tuple.

3.2 Compiler Input and Attribute Resolution

The slicer project compiler consumes heterogeneous implicit designs produced by an upstream attribute-modeling workflow [Wade et al. 2026]. Each input design provides an implicit solid geometry together with named spatial attribute fields defined over the object. These attributes may describe slicer-planning settings, process-state setpoints, color, material fractions, or higher-level design intent.

The selected slicer-project target determines which spatial fields must be available before project generation. Settings-mesh targets consume slicer-planning attributes such as infill density or fuzzy-skin parameters. Virtual-extrusion targets consume process-state attributes such as nozzle temperature and flow-rate compensation. Color and material halftoning targets consume color fields or material-fraction fields. Our proposed compiler then lowers these resolved fields into slicer-native project assignments.

Consistent with heterogeneous implicit design methods [Wade et al. 2026], required process fields may be authored directly or be resolved from higher-level intent before compilation. For example, a design may specify target density or Shore hardness rather than nozzle temperature. In addition to our slicer compiler, we introduce new translation models to resolve printer process parameters from design intent. As we show in section 4.2.1, our calibrated translation models can compute the temperature and flowrate fields for fabrication with foaming filaments.

This separation between translation models and slicer-project compilation keeps the compiler target-specific but design-source agnostic. The same compiler can consume directly authored slicer settings, translated process fields, or color and material-fraction fields, provided that the selected target receives the attributes it requires. The remaining methods therefore focus on how those resolved fields are partitioned, converted into sub-mesh regions, and serialized into slicer-native `.3mf` projects.

3.3 Slicer-Native `.3mf` Projects as Compiler Outputs

A slicer-native `.3mf` project records the fabrication setup that a slicer normally builds through interactive use. In a conventional workflow, a designer imports one or more meshes, transforms and arranges them on the build plate, selects global print settings, applies model- or region-level overrides, and assigns materials or extruders. The slicer can then save this configured job as a single project file for later editing, sharing, or slicing. The project therefore contains more than boundary geometry: it stores the relationship between geometric objects and the slicer state needed to fabricate them.

This makes a slicer-native `.3mf` project a useful compiler target. A conventional mesh file can describe surfaces, but it cannot express the slicer settings, tool assignments, or mixed-filament recipes required by a heterogeneous design. A project file can carry this missing context in a form the slicer interprets. The slicer project compiler therefore automates the interactive setup process: it generates the required sub-meshes, attaches the appropriate per-region assignments, writes the project-level configuration, and emits a configured fabrication job rather than an unannotated collection of meshes.

Different slicers serialize these project components using different dialects. PrusaSlicer-style projects and Orca/Bambu component-style projects both store geometry, project settings, and per-region assignments, but they organize those data differently and use different metadata conventions. The slicer project compiler separates the abstract assignment produced by the method from the dialect-specific serialization written to disk.

3.4 Attribute Partitioning

As detailed in Figure 2, all three slicer project targets use the same geometric reduction: the compiler converts continuous attribute fields over the object interior into a finite set of labeled sub-meshes. The compiler samples the implicit geometry and the required attributes on a regular grid and assigns sampled points to discrete attribute regions, where a region represents a range of attribute values. The compiler then constructs a region-indicator mask for each region, intersects that mask with the object domain, and extracts the resulting boundary with marching cubes [Lorensen and Cline 1987]. The output is a collection of sub-meshes whose labels form a piecewise-constant approximation of the resolved functionally graded attributes. Each sub-mesh is assigned a single attribute value.

The compiler extracts each region as a sub-object. For a selected attribute bin or recipe, it defines a region-membership field whose

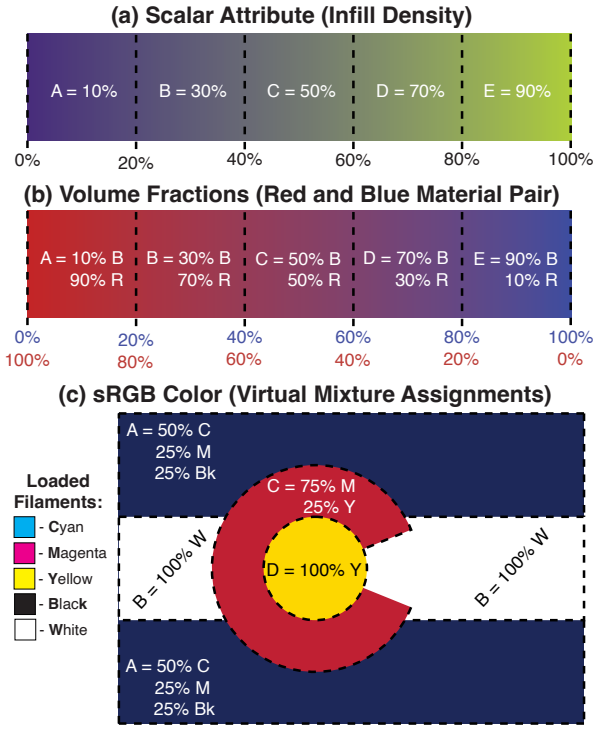


Figure 2: Attribute partitioning converts continuous heterogeneous fields into finite slicer-project assignments. (a) Scalar fields are divided into representative intervals, shown here for infill density. (b) Material-fraction fields are partitioned in recipe space so each region carries a valid mixture vector. (c) Color fields are mapped to selected virtual mixture assignments derived from the loaded filament set.

zero set marks the boundary between points assigned to that region and points assigned elsewhere. The compiler combines this field with the object SDF so that the extracted surface is clipped to the original solid domain. Marching cubes then extracts the boundary of this sub-object as a triangle surface mesh. As a result, the region meshes can place boundaries between grid samples when the geometry and attribute fields vary smoothly, even though the compiler samples the fields on a regular grid. The chosen region count sets the fidelity–cost tradeoff: more regions better preserve spatial variation, but produce more sub-meshes, larger project files, and more downstream slicer work.

3.4.1 Scalar Attribute Partitioning. Scalar partitioning supports settings meshes, virtual extrusion, and any workflow that must assign one or more scalar controls jointly. Given scalar fields, the compiler first measures each field's observed range over sampled points inside the object. It then divides each range into N equal-width intervals and uses each interval midpoint as the representative value assigned to regions in that interval.

When several scalar attributes must vary together, the compiler constructs the Cartesian product of their intervals. A point belongs to a product cell only if every participating attribute falls within

the corresponding interval. This joint partition ensures that each generated sub-mesh carries an identical tuple of representative values, such as a temperature and flow-rate pair or a fuzzy-skin setting together with a virtual-extrusion state. With m attributes and N intervals per attribute, the method can produce up to N^m candidate regions.

Algorithm 1 summarizes this process. For settings meshes, the representative labels become slicer-setting values. For virtual extrusion, they become logical process-state assignments. For combined workflows, the labels may include both planning settings and process-state values.

Algorithm 1. Partition an Implicit Object by Scalar Attribute Fields

Input: Implicit geometry; scalar attribute fields; intervals N per attribute
Output: Labeled sub-mesh regions with representative scalar values

```
Sample the geometry and scalar attributes inside the object;
foreach attribute field A_i do
    Compute its sampled value range;
    Partition the range into N equal-width intervals;
    Store each interval midpoint as its representative value;
Form the Cartesian product of intervals across all attributes;
foreach interval tuple B do
    Select the object region whose attribute values fall in B;
    Intersect the selected region with the implicit geometry;
    Extract its boundary as a sub-mesh;
    if the sub-mesh is nonempty then
        Label it with the representative values from B;
        Add it to the output;
Return the labeled sub-meshes;
```

3.4.2 Material-Fraction Partitioning. Material-fraction fields require a different value-space partition because each sample is a recipe vector rather than an independent scalar. For k materials, each sampled material-fraction value is a nonnegative length-k vector whose components sum to one. Component-wise scalar binning would create many invalid or redundant recipe combinations, so the compiler partitions the sampled vectors directly in material-fraction space.

The compiler divides material-fraction space into a bounded set of representative recipes. Unlike an arbitrary scalar field, a material-fraction field has known bounds: each component lies between 0 and 1, and the components form a normalized fraction vector. The compiler can therefore partition the full feasible recipe space directly. For a two-material field ($k = 2$) with materials A and B, choosing $N = 4$ creates four fraction ranges: 0–25% A with 100–75% B, 25–50% A with 75–50% B, 50–75% A with 50–25% B, and 75–100% A with 25–0% B. The compiler assigns each sampled material-fraction vector to the nearest range, computes a representative fraction vector for each occupied range, and maps that vector to a printable recipe. The resulting spatial regions are then extracted with the same geometry-intersection and marching-cubes procedure used for scalar fields.

Algorithm 2 describes this procedure. The key distinction from scalar partitioning is that the partition is built in the attribute's natural value space: scalar intervals for scalar fields, and median-cut recipe regions for material-fraction vectors.

Algorithm 2. Extend Attribute Partitioning to Material Volume Fractions

Input: Implicit geometry; material-fraction field; maximum regions N
Output: Labeled sub-mesh regions with representative material recipes

```
Sample the material-fraction field inside the object and normalize each vector;
Partition material-fraction space into N groups;
foreach group G_j do
    Compute a representative fraction vector v̄_j;
Assign each object point to the nearest representative v̄_j;
foreach representative v̄_j do
    Form the spatial region assigned to v̄_j;
    Extract its nonempty sub-mesh using Algorithm 1;
    Label the sub-mesh with material recipe v̄_j;
Return the labeled material-recipe sub-meshes;
```

3.4.3 Color-Field Partitioning and Palette Selection. Color fields require an additional selection step because the loaded filaments cannot reproduce every sRGB value equally well. If the compiler assigned each sampled color independently, it could request many recipes whose predicted colors are poor approximations or whose small differences provide little visible benefit. We therefore select a bounded printable palette before spatial region extraction. The palette contains recipes that cover the colors present in the object, given the fixed physical filaments and the target slicer's color-mixing model.

As shown in Figure 3, the input is an sRGB color field and a fixed set of loaded physical filaments with specified RGB colors. The compiler internally generates a discrete vocabulary of printable recipes containing up to three filaments at fixed proportions. The downstream slicers limit recipes to three filaments. Workflow-specific color-prediction models estimate the apparent color of each candidate recipe. In this paper, the ColorMix target uses `prusa-fdm-mixer`, while the FullSpectrum target uses `filament-mixer`. These models predict recipe appearance; they do not generate the recipe vocabulary.

The compiler aggregates sampled target colors into a weighted distribution and greedily selects at most N candidate recipes that minimize sample-count-weighted CIEDE2000 error, ΔE_{00}. Each sampled point is assigned to the selected recipe with the smallest predicted ΔE_{00} from the requested color $C(p)$. These assignments define color-recipe regions, which are extracted as sub-meshes using

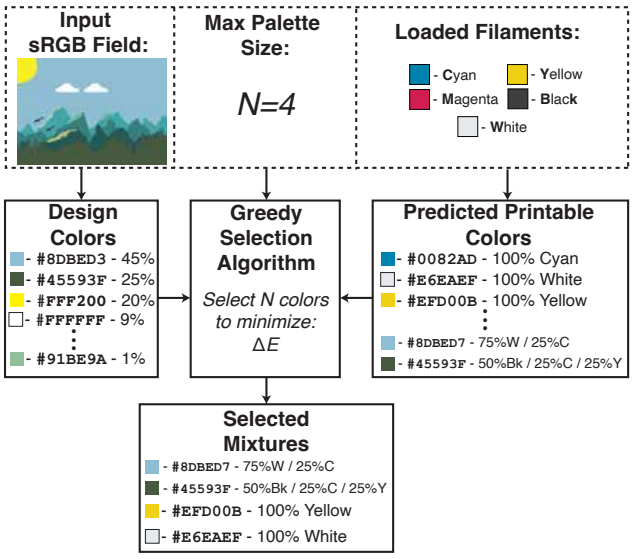


Figure 3: Palette selection for compiling an sRGB color field into printable mixture assignments. The compiler samples the design colors, predicts the apparent colors of candidate recipes from the loaded filaments, and greedily selects at most N mixtures that minimize weighted perceptual color error. The selected mixtures become the labels used for downstream color-region extraction and project serialization.

Algorithm 3. Compile a Full-Color Attribute Field into a Slicer Project

Input: Implicit geometry; sRGB color field; filament palette; candidate recipes; target workflow; maximum palette size N
Output: Color-assigned sub-meshes in a 3MF project and a color report

Sample C inside the object and aggregate similar colors into a frequency distribution;
foreach *candidate recipe* **do**
  Predict its printable color;
Greedily select at most N recipes that minimize frequency-weighted perceptual color error;
foreach *sampled point p inside the object* **do**
  Assign its color to the selected recipe with the closest predicted color;
foreach *selected recipe* **do**
  Form the spatial region assigned to that recipe;
  Extract its nonempty sub-mesh using Algorithm 1;
  Label the sub-mesh with the corresponding filament or recipe;
Package filament definitions, printable mixtures, color-assigned sub-meshes, and slicer metadata into a 3MF project;
Return the 3MF project and color report;

the same geometric procedure as the scalar and material-fraction cases.

Algorithm 3 summarizes full-color slicer project compilation. The algorithm differs from the previous two only in how it chooses the representative labels. Once the selected printable recipes exist, the spatial extraction step again produces labeled sub-meshes that can be serialized into the target slicer-project dialect.

3.5 Slicer-Planning Control with Settings Meshes

Settings meshes provide direct control over slicer-planning parameters by encoding partitioned scalar attributes as slicer-recognized regions with setting overrides. These meshes are generated directly from the partitioning process described in Algorithm 1. In the emitted `.3mf` project, each region is represented as XML metadata in the slicer's native dialect. Figure 4a shows a representative example for PrusaSlicer in which a region mesh is assigned an infill-density override. Because these values exist before toolpath generation, they can change the slicer's planning behavior rather than post-process the generated G-code.

Settings meshes only apply to parameters that the slicer already exposes as model- or volume-level overrides. Examples include infill density, fuzzy skin parameters, speeds, extrusion width, and perimeter count. This component of our workflow automates a process that would otherwise require manual project construction. Without slicer project compilation, a designer must export separate region meshes, import them into the slicer, align them, and assign each setting override through the user interface. The proposed compiler instead emits one configured `.3mf` project in which the region geometry and settings assignments are embedded. We report the number of user interactions saved by using our proposed method compared to manual assignment in Section 4.4.

3.6 Machine-State Control with Virtual Extrusion

Settings meshes control parameters that the slicer already exposes as per-region planning settings. Virtual extrusion addresses a different case: spatially varying process states that the slicer does not ordinarily expose as local model settings. We represent each discrete process-state tuple as a logical tool. The slicer then emits toolchange events between those logical tools, and a custom toolchange template converts each event into G-code commands that update machine state.

As shown in Figure 5, virtual extrusion separates physical extruders from virtual extruders. A physical extruder is an actual printer tool, nozzle, or filament channel. A virtual extruder is a logical tool index used by the slicer to identify a process-state setpoint or setpoint tuple. The printer may still have one physical nozzle, but the project can declare additional logical tools whose toolchange events map to commands such as temperature or flow-rate updates. In this paper, we demonstrate this mechanism with

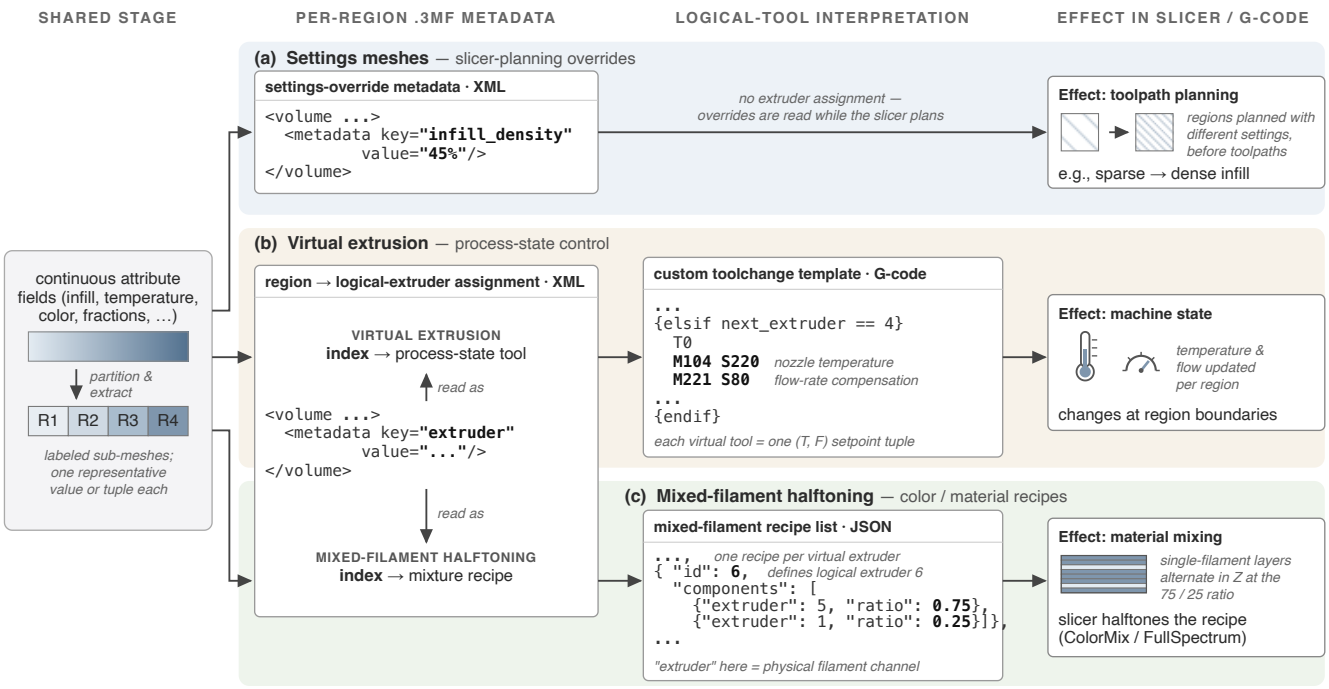


Figure 4: The three slicer-project compilation targets write different per-region assignments from the same partitioned sub-meshes. Settings meshes (a) attach slicer-planning overrides directly to region metadata and involve no tool assignment. Virtual extrusion (b) and mixed-filament halftoning (c) share the region-to-logical-extruder assignment but interpret the extruder index differently: (b) resolves it through a custom toolchange template into process-state G-code such as `M104` (temperature) and `M221` (flow rate), while (c) resolves it to an entry in a recipe list that the slicer halftones from the loaded physical filaments. Excerpts are abbreviated; the compiler writes the full native dialect of the target slicer.

temperature and flow rate, using `M104` to set nozzle temperature and `M221` to set flow-rate compensation. However, the same mechanism can emit any G-code sequence that the printer accepts.

The compiler applies Algorithm 1 to the process-state fields. Each occupied scalar interval, or joint interval for multiple process-state fields, receives a representative tuple such as (T, F), where T is temperature and F is flow-rate compensation. The compiler assigns each resulting sub-mesh to the logical tool associated with that tuple. The slicer sees these regions as ordinary tool assignments, but the custom toolchange template interprets the next logical tool index as a request to update machine state. Figure 4b shows this composition: each region carries a logical-extruder assignment, and the template converts the resulting toolchange events into the corresponding process-state commands.

This mechanism is useful for foaming filaments because the local expansion state depends on nozzle temperature, while geometric accuracy often requires compensatory flow control. A pre-compilation translation model can therefore resolve a desired density or hardness field into temperature and flow-rate fields, and virtual extrusion can then express those resolved process fields in a slicer project without requiring a custom slicer or custom path generator. Table 1 summarizes the distinction between settings meshes and virtual extrusion.

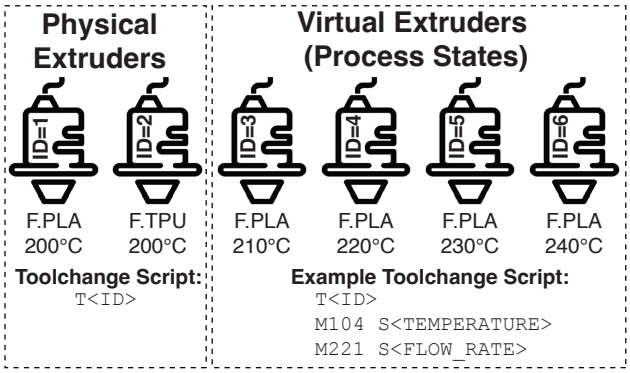


Figure 5: Virtual extrusion maps logical slicer tools to process-state setpoints rather than only physical extruders. Physical extruders correspond to real filament channels, while additional virtual extruders encode states such as nozzle temperature and flow-rate compensation. Custom toolchange scripts (Fig. 4b) convert slicer toolchange events into the G-code commands required to apply each process state.

Table 1: Settings meshes and virtual extrusion both use scalar partitioning, but they affect different stages of the fabrication workflow.

Method	Controls	When applied
Settings meshes	Slicer planning parameters, such as infill, speeds, widths, and fuzzy skin.	Before toolpath generation; the slicer uses the values during planning.
Virtual extrusion	G-code-controllable process states, such as temperature and flow rate.	At logical toolchange events in the generated G-code.

Ordering virtual tools for slow process states. Some virtual extrusion workflows also require an ordering policy. This is not necessary for every G-code-controllable state, but it matters when transitions have time or material costs. Temperature control for foaming filaments is one such case: the nozzle needs time to reach a new temperature, and the extruded material may require purging or a prime tower before the new foaming state stabilizes.

For temperature-based virtual extrusion, the compiler indexes virtual tools in increasing temperature. We then use a reversing layer order: the slicer visits tools in ascending order on one layer and descending order on the next, for example 1, 2, 3, 4, 5 followed by 5, 4, 3, 2, 1. This preserves monotonic local changes while avoiding a large reset from the hottest state to the coolest state at each layer boundary.

3.7 Filament-Mixture Control with Color and Material Halftoning Projects

ColorMix and FullSpectrum-style slicer workflows provide a third form of slicer project control: they approximate intermediate colors or material recipes by assigning regions to virtual mixed-filament recipes. A printer loads a fixed set of physical filaments, such as CMYKW, and the slicer defines virtual recipes that combine one, two, or three of those filaments at specified ratios. The slicer then performs the downstream halftoning or filament-mixing procedure needed to realize those recipes. The slicer project compiler uses this mechanism by generating recipe definitions and assigning extracted sub-meshes to the corresponding virtual recipes.

This workflow supports two input cases. If the resolved design contains a material-fraction field, the compiler uses Algorithm 2 directly. It partitions the field into representative material-fraction regions, maps each component of the volume-fraction vector to a loaded filament channel, and writes the project metadata that defines the mixture ratio for each generated sub-mesh. This path bypasses color matching because the design already specifies the desired material mixture rather than an appearance target.

If the resolved design instead contains an RGB color field, the compiler treats the loaded filament set as a constrained color gamut. It does not assume that arbitrary sRGB values can be reproduced accurately. Instead, as described in Algorithm 3, the compiler generates a discrete vocabulary of printable recipes, predicts each recipe's apparent color with the workflow-specific color model, and selects a bounded palette that best covers the sampled target colors under a sample-count-weighted ΔE_{00} objective. It then assigns each spatial region to the selected recipe with the smallest predicted perceptual error.

Prusa ColorMix and Orca FullSpectrum use the same abstract representation of physical filaments, virtual mixtures, and region assignments, but they serialize those data differently and use different downstream half-toning algorithms. The compiler therefore treats them as separate project dialects under one method: both consume labeled sub-meshes and recipe definitions, but each requires its own native metadata. Figure 4c shows a representative entry in the generated recipe list.

The virtual recipes used in this section are distinct from the logical tools used for virtual extrusion. Both mechanisms use slicer-level tool identifiers, but their interpretations differ (Fig. 4): halftoning projects use virtual recipes to control filament mixtures, while virtual extrusion uses logical tools to trigger process-state changes.

4 Results

The results evaluate the compiler as a bridge between heterogeneous design intent and slicer-native fabrication workflows. We first show that scalar attributes can control slicer settings through generated settings meshes, allowing the downstream slicer to plan different toolpaths across an object without manual region assignment. We then evaluate virtual extrusion as a mechanism for process-state control, using calibrated foaming-filament models to translate density and Shore-hardness fields into temperature and flow-rate assignments. Next, we combine slicer-setting and process-state controls in a single compiled project to test whether the same partitioning framework can coordinate multiple fabrication effects. We then evaluate color and material halftoning compilation by converting material-fraction and sRGB fields into ColorMix and FullSpectrum-compatible projects and measuring how the downstream slicing workflows reproduce color gradients. Finally, we quantify the reduction in user interactions obtained by emitting configured `.3mf` projects directly.

4.1 Slicer-Setting Control with Generated Settings Meshes

First, we evaluate the translation to settings meshes: whether the compiler can translate scalar heterogeneous attributes into slicer-planning controls rather than encoded geometry. In these examples, the source design contains an infill density or fuzzy-skin attribute field. The compiler partitions the field, extracts labeled sub-meshes, and writes a single PrusaSlicer-compatible `.3mf` project in which each region carries the corresponding slicer setting. We slice each project in PrusaSlicer 2.9.6 and print the examples on a Prusa Core One+ using PLA, a 0.4 mm nozzle, and 0.2 mm layers. For the infill examples, we disable top and bottom skins so that the printed artifacts expose the internal infill structure.

4.1.1 Infill Density. Figure 6 shows two compiled infill-density fields. The rectangular prism defines a linear infill density gradient from 5% to 80% along the x-axis. We set the partition count to $N = 12$, and the compiler discretizes the scalar field into 12 intervals and outputs 12 aligned sub-meshes. PrusaSlicer then plans each region with the assigned infill density. The printed artifact

```
bar_size_x = 100.0
bar_size_y = 25.0
bar_size_z = 25.0

infill_expr = (
    f"5 + 75 * ((x + {bar_size_x / 2}) / {bar_size_x})"
)

bar = pv.RectPrism(
    pv.Vec3(0, 0, 0),
    pv.Vec3(bar_size_x, bar_size_y, bar_size_z),
)
bar.set_attribute(
    pv.DefaultAttributes.INFILL_DENSITY,
    pv.FloatAttribute(infill_expr),
)
```

Listing 1: Source definition for the linear infill-density bar in Fig. 6a.

exposes the expected monotonic change in cell density along the designed axis, showing that the slicer interprets the generated region settings as ordinary local planning controls.

The bracket example tests the same compilation mechanism on an application-motivated design. The source field assigns high infill density near the three mounting holes and low infill density elsewhere, then blends the transition over a 13 mm radius to produce a smooth field over the bracket. The implicit design specifies an infill-density field ranging from 10% to 70%. We set the partition count to $N = 6$, and the compiler discretizes this range into six infill regions. In the sliced preview and printed artifact, the densest infill appears around the mounting holes and decreases away from them. This result demonstrates a practical use of settings meshes: the designer specifies where increased local stiffness is desired, while the downstream slicer still supplies its native infill pattern, toolpath ordering, and printer-specific planning.

4.1.2 Fuzzy Skin. Figure 8 visualizes our evaluation of settings meshes for two fuzzy-skin parameters. As shown in Figure 7, PrusaSlicer exposes two settings to control fuzzy skin: point thickness and point distance. Fuzzy-skin thickness sets the maximum outward perturbation from the nominal perimeter path, while fuzzy-skin point distance sets the spacing between successive perturbation points. Larger thickness increases the amplitude of the surface texture, and larger point distance produces a coarser texture.

The source design assigns fuzzy-skin thickness as a linear field from 0.0 mm to 1.0 mm along the vertical axis and fuzzy-skin point distance as a separate linear field from 0.5 mm to 1.5 mm along the horizontal axis. The compiler partitions these fields into regular setting regions and writes the corresponding per-region fuzzy-skin overrides. The printed swatches show the expected independent effects: increasing thickness increases roughness amplitude, while increasing point distance produces a coarser and less dense texture. These examples show that settings meshes can control multiple slicer-exposed parameters as spatial gradients without redefining the surface texture as geometry.

We next apply fuzzy-skin settings to a more complex implicit object. Figure 9a defines semantic regions for the Stanford bunny, including the body, feet, ears, tail, and eyes. These regions are volumetric attributes, not surface-only paint strokes. We highlight this distinction because we use the same implicit volumetric design for

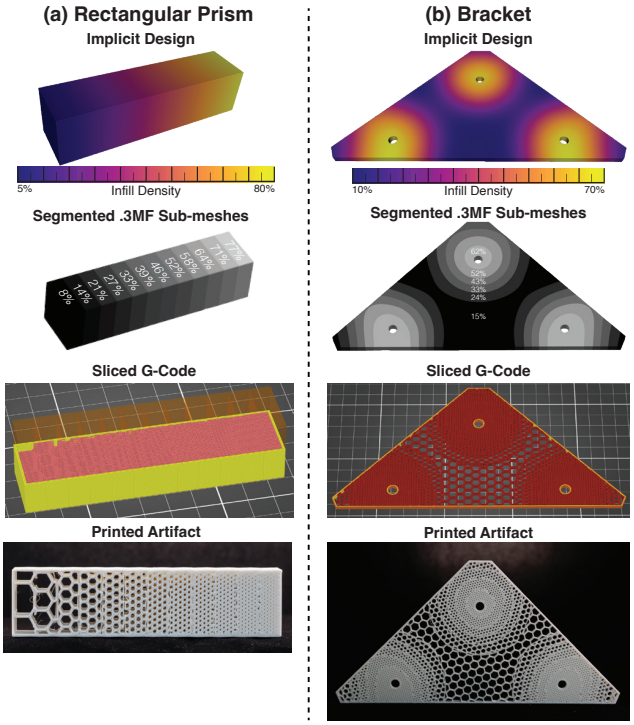


Figure 6: Generated settings meshes for spatially varying infill density. (a) A rectangular prism defines a linear infill-density field from 5% to 80%, which the compiler partitions into 12 sub-meshes and exports as a configured `.3mf` project. (b) A bracket defines high infill near three mounting holes and lower infill elsewhere, blended over the object and partitioned into six regions. In both examples, PrusaSlicer plans the local infill from the generated region settings.

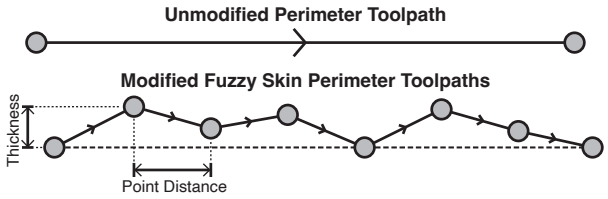


Figure 7: Fuzzy-skin parameters exposed by the downstream slicer. Thickness controls the maximum offset from the nominal perimeter, while point distance controls the spacing between perturbation points along the perimeter.

this example of surface settings and an example in Sec. 4.2 that modulates shore hardness over the volume. In this subsection, we use only the fuzzy-skin thickness and point-distance fields shown in Fig. 9c–d. The ears, eyes, and feet receive no fuzzy-skin thickness, while the body and tail receive larger thickness values. The body uses a smaller point distance to create a dense fuzzy texture, and the tail uses a larger point distance to produce a coarser,

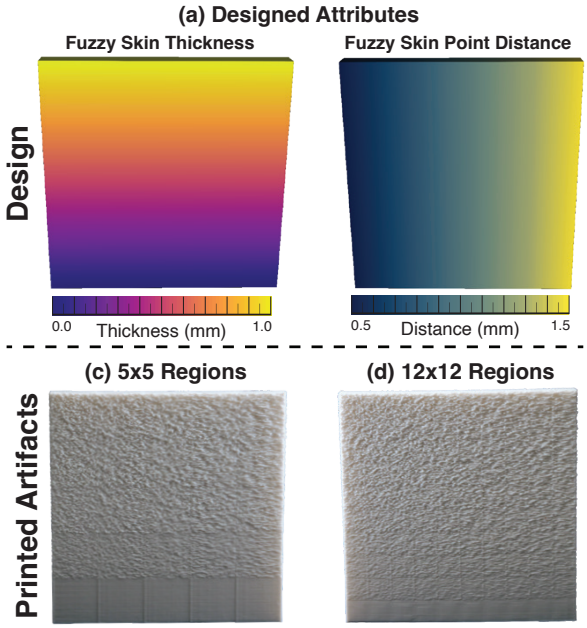


Figure 8: Generated settings meshes for fuzzy-skin parameters. The source design varies fuzzy-skin thickness from 0.0 mm **to** 1.0 mm **and fuzzy-skin point distance from** 0.5 mm **to** 1.5 mm**. The printed swatches show that the compiler can assign these slicer settings independently across generated regions.**

Table 2: Semantic attribute values assigned to the bunny. This subsection evaluates the fuzzy-skin settings; Sec. 4.2 evaluates the Shore-hardness field using the same semantic design.

Region	Thickness (mm)	Point dist. (mm)	Shore A
Feet	0.0	1.0	85
Ears	0.0	1.0	65
Tail	0.5	1.0	65
Eyes	0.0	1.0	65
Body	0.5	0.5	85

bushier texture. Table 2 lists the region values used for both the fuzzy-skin settings and the Shore-hardness field evaluated later.

Blending converts the piecewise semantic regions into smooth spatial setting fields, as shown in Fig. 9b–d. We set the partition count to $N = 12$ for both fuzzy-skin thickness and point distance. The compiler constructs the resulting 12×12 joint partition, producing 144 setting combinations before removing empty regions.

Figure 10 compares printed bunnies compiled with and without blended setting fields. The unblended print has abrupt texture transitions where regions change from fuzzy to smooth. The blended print removes these visible boundary artifacts and produces a gradient change in texture across the body, legs, and tail. This result shows that the compiler can preserve region structure when sharp

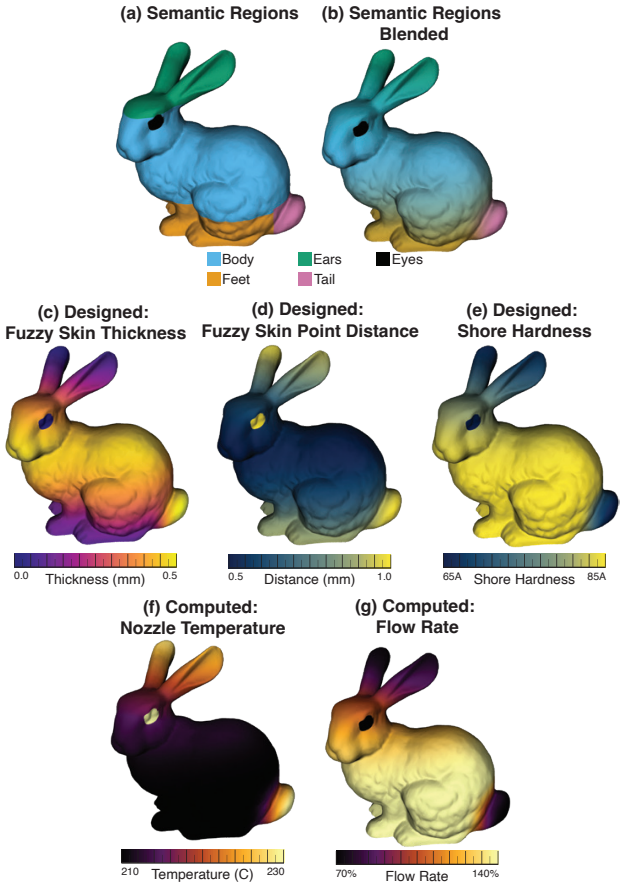


Figure 9: Volumetric semantic attributes for the fuzzy-skin bunny. (a) Semantic regions assign settings to anatomical regions of the bunny. (b) A design-stage blending operation, performed before compilation and slicing, converts the piecewise semantic assignments into smooth volumetric fields. (c,d) The resulting fuzzy-skin thickness and point-distance fields drive settings-mesh compilation. Panels (e–g) define the Shore-hardness, temperature, and flow-rate fields evaluated later for virtual extrusion.

transitions are desired, or apply blended volumetric attributes when smoother setting variation is required.

These examples establish settings meshes as a direct compilation target for slicer-planning parameters. The compiler does not replace the slicer's infill generator or fuzzy-skin toolpath logic. Instead, it converts heterogeneous attributes into the local settings that the slicer already understands. The next section evaluates the complementary case, where a spatial field controls machine state through virtual extrusion rather than slicer planning.

4.2 Process-State Control with Virtual Extrusion

Next, we evaluate virtual extrusion: whether the compiler can control machine state through slicer-native tool assignments. In this

Figure 10: Printed fuzzy-skin bunnies compiled from semantic setting fields. The unblended bunny has sharp texture transitions at semantic-region boundaries, while the blended bunny uses a joint 12×12 settings partition to produce smoother texture variation.

workflow, the compiler partitions process-state fields into discrete regions, maps each region to a logical tool, and relies on the slicer's toolchange events to output the corresponding G-code commands. We demonstrate this mechanism with temperature responsive foaming filaments, where nozzle temperature controls expansion, density, and Shore hardness, and flow-rate compensation preserves geometry. All examples in this section are sliced with PrusaSlicer 2.9.6 and printed by a Prusa MK4S.

4.2.1 Foaming-Filament Calibration. We first characterize the process models needed to use foaming filaments as controllable attribute-translation targets. We evaluate ColorFabb VarioShore TPU and ColorFabb LW-PLA by printing single-wall calibration samples over a range of nozzle temperatures. Each sample uses a designed wall thickness of 0.45 mm, corresponding to one wall with a 0.4 mm nozzle, and repeats that wall for 20 mm in height. For each temperature, we print four samples, measure each wall with a micrometer, and report the mean wall thickness with one standard deviation.

Figure 11a shows uncompensated wall expansion as a function of nozzle temperature. Both materials expand as temperature increases. VarioShore TPU begins expanding near 208°C and reaches approximately 0.97 mm wall thickness at high temperature, corresponding to about 216% of the designed wall thickness. LW-PLA begins expanding at a higher temperature, near 222°C, and reaches approximately 0.85 mm, or about 188% of the designed wall thickness. At low temperatures, both materials print slightly below the designed wall thickness, so the compensation model must correct both under-extrusion and over-expansion.

We compute the flow-rate multiplier required at each temperature from the measured wall expansion:

$$\lambda(T) = \alpha \frac{W_D}{W(T)}, \tag{1}$$

where $\lambda(T)$ is the flow-rate multiplier at temperature T, $\alpha = 1.0$ is the base flow-rate multiplier, $W_D = 0.45$ mm is the designed wall thickness, and $W(T)$ is the average measured wall thickness at temperature T. Figure 11b plots the resulting compensation model for both materials. The compiler interpolates the measured calibration data to evaluate $\lambda(T)$ and converts the resulting multiplier to the percentage form required by the emitted G-code command.

Figure 11c validates the compensation model. After applying the temperature-dependent flow-rate multiplier, both TPU and LW-PLA remain close to the 0.45 mm target despite large changes in foaming expansion. This result establishes the process-state pair used by virtual extrusion: temperature controls the material response, while flow-rate compensation preserves bead geometry.

We then use the compensated process to characterize density. We print 30 mm × 30 mm × 7 mm solid blocks at 100% infill with normal top and bottom skins, apply the calibrated flow-rate compensation, measure the mass and external dimensions, and compute density from mass divided by measured external volume. Figure 11d shows that density decreases monotonically with temperature for both materials. This relationship defines an inverse translation model from target density to nozzle temperature, followed by the temperature-to-flow-rate model in Eq. 1. This calibration defines an inverse translation model: target density maps to nozzle temperature, and temperature then maps to flow-rate compensation.

VarioShore TPU also changes Shore A hardness with temperature. We characterize this relationship by printing 30 mm×30 mm× 7 mm samples in compliance with ASTM D2240 and measuring Shore A hardness with a durometer. Figure 11e shows that hardness decreases as temperature increases, consistent with the increased foaming and reduced density observed in Fig. 11d. This

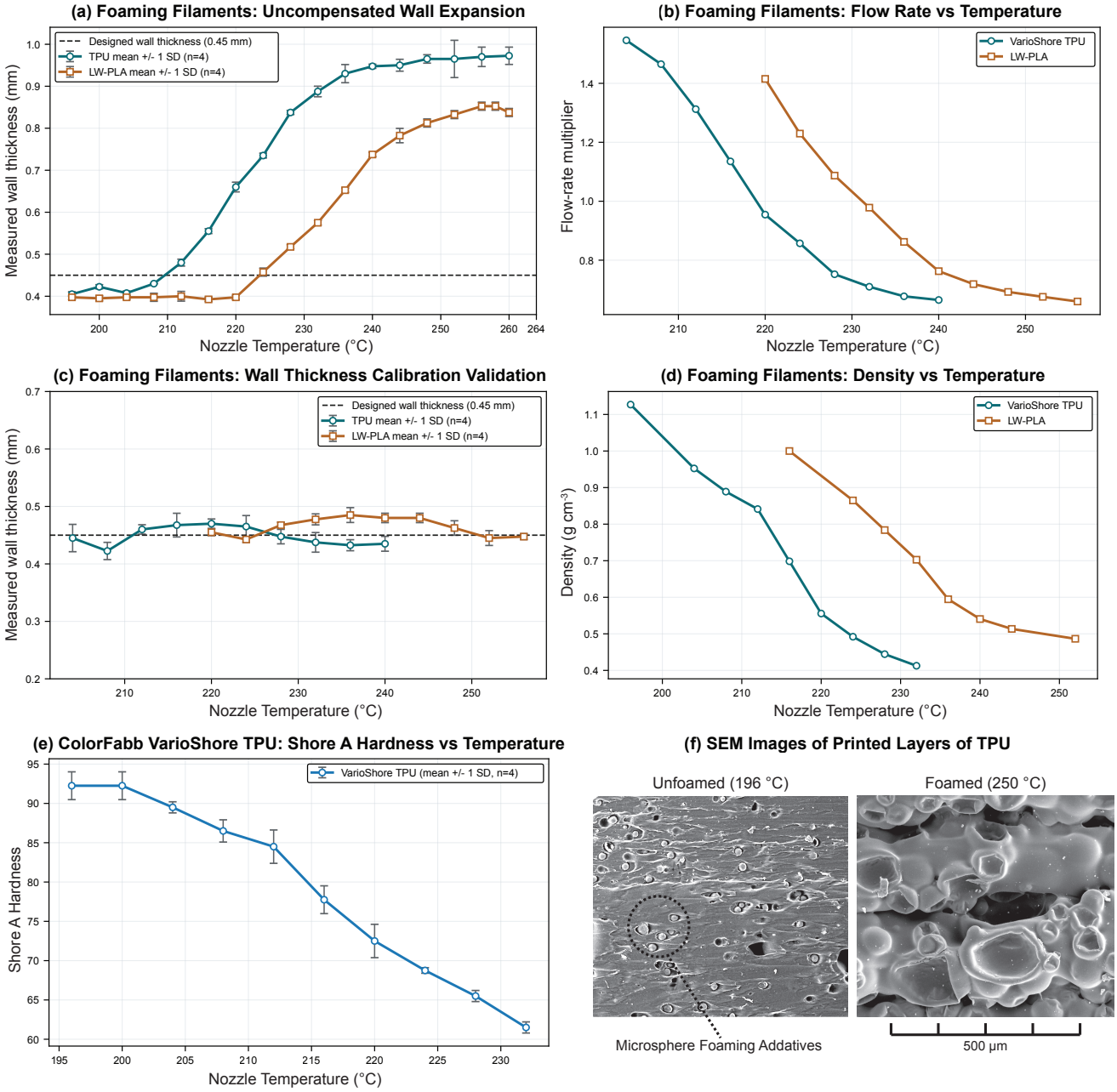


Figure 11: Calibration data for temperature responsive foaming filaments. (a) Uncompensated wall expansion for VarioShore TPU and LW-PLA. (b) Flow-rate multiplier computed from measured wall expansion. (c) Compensated wall-thickness validation, showing measured wall thickness near the 0.45 mm target across temperature. (d) Density as a function of temperature for compensated solid samples. (e) Shore A hardness as a function of temperature for VarioShore TPU. (f) SEM images of unfoamed and foamed TPU, showing the transition from intact microsphere additives to gas-filled pores.

calibration defines another inverse translation model: target Shore hardness maps to nozzle temperature, and temperature then maps to flow-rate compensation.

Figure 11f provides a physical interpretation of these trends. At 196˚C, the TPU remains largely unfoamed and the microsphere additives remain visible. At 250˚C, the microspheres expand into gas-filled pores within the printed material. This change in pore

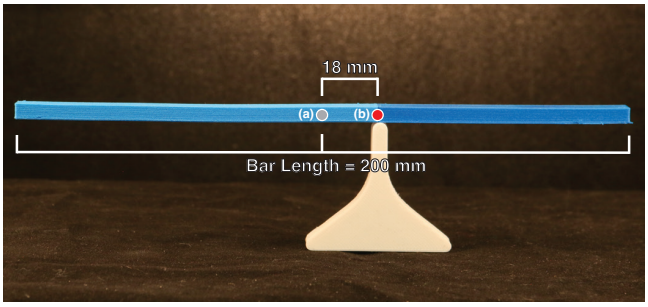


Figure 12: Density inverse design with foaming TPU. The bar is 200 mm **long and is designed to place its center of mass (b)** 18 mm **to the right of the geometric center (a). The printed artifact balances at the designed offset, showing that the compiled temperature and flow-rate fields realize the intended density distribution.**

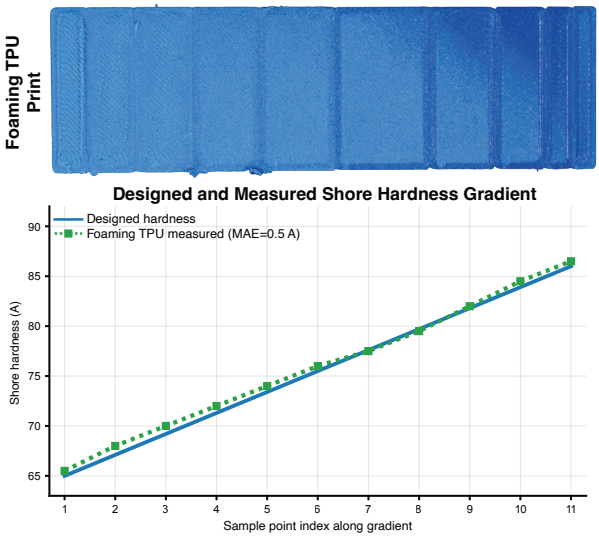


Figure 13: Shore-hardness inverse design with foaming TPU. The source design specifies a linear 65A–85A hardness gradient. The measured values along the printed bar follow the designed gradient with a mean absolute error of 0.5 **Shore A.**

structure explains why increasing temperature lowers both density and Shore A hardness, and why temperature can serve as the process-state variable for inverse design.

4.2.2 Density Inverse Design. Figure 12 demonstrates density-based inverse design with foaming TPU. The source design is a 200 mm-long bar whose density field is chosen to place the center of mass 18 mm to the right of the geometric center. The design uses two calibrated density states from Fig. 11d: a low-density foamed state of 0.413 g cm^{-3} and a high-density unfoamed state of 1.123 g cm^{-3}. The solver selects the density switch point required to achieve the requested offset.

The printed bar balances at the designed off-center location. We determine the realized center of mass by finding the point along the printed artifact where it remains balanced on a support and measuring that point relative to the geometric center. The observed balance point confirms that the density field compiles into a physically meaningful mass distribution. The user specifies the high-level density objective, and the attribute-translation models derive the temperature and flow-rate fields that virtual extrusion applies during slicing and printing.

4.2.3 Shore-Hardness Inverse Design. Figure 13 validates the Shore-hardness inverse model from Fig. 11e on a gradient bar. The source design specifies a linear Shore-hardness field from 65A to 85A. Before compilation, the translation model computes the required temperature field, and the flow-rate compensation model in Eq. 1 computes the companion flow-rate field. Virtual extrusion then assigns the resulting process-state tuples to logical tools in a PrusaSlicer project. We measure hardness at 11 evenly spaced points along the printed bar and compare the measurements with the designed values. The measured gradient follows the target field with a mean absolute error of 0.5 Shore A, indicating that the compiled process-state assignments preserve the intended mechanical-property gradient over the sampled locations.

4.2.4 Foaming Fuzzy Bunny. The foaming fuzzy bunny combines process-state control with the settings-mesh control evaluated in Sec. 4.1. The semantic design is the same one summarized in Table 2 and visualized in Fig. 9. The body and feet use the higher hardness value of 85A, while the tail, ears, and eyes use the softer 65A value. The fuzzy-skin thickness and point-distance fields remain active, so the compiler must coordinate two slicer-setting fields with one process-state field.

We set the partition count to $N = 6$ for fuzzy-skin thickness, fuzzy-skin point distance, and nozzle temperature. The compiler then discretizes the three fields into a $6 \times 6 \times 6$ product space. This example simultaneously exercises multiple compilation targets: a single volumetric design produces a slicer-ready project containing local surface texture settings and local machine-state control. The interaction savings for this joint design are quantified later in Sec. 4.4.

Figure 14 shows the printed artifact. The surface texture follows the fuzzy-skin fields, while the subtle color shift follows the temperature and foaming field. Lighter regions correspond to higher-temperature, lower-density, softer foamed TPU; darker regions correspond to lower-temperature, higher-density, harder TPU. The eye region makes this contrast visible because it preserves a sharp transition. This result shows that virtual extrusion can operate together with settings meshes in a single downstream slicer workflow, allowing high-level volumetric attributes to control both process state and slicer-planned surface texture.

4.3 Color and Material-Fraction Control with Halftoning Filament Mixing

Finally, we evaluate halftone filament mixing: whether the compiler can translate material-fraction and color attributes into slicer-ready projects for downstream color-mixing workflows. In these examples, the printer is a Prusa XL with five PLA filaments loaded

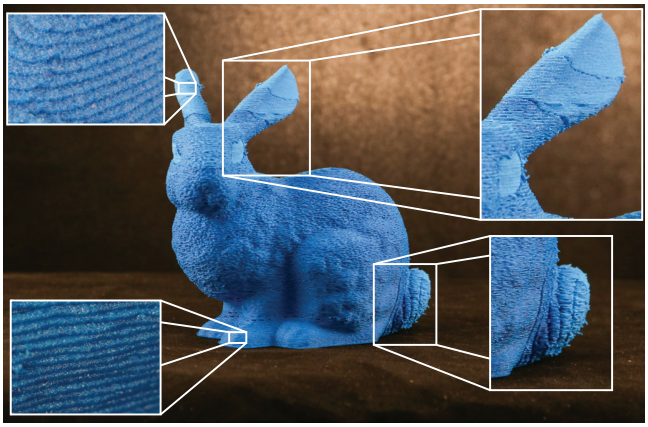


Figure 14: Foaming fuzzy bunny printed from a compiled PrusaSlicer project. The object combines fuzzy-skin settings with virtual-extrusion control of nozzle temperature and flow-rate compensation. Lighter regions correspond to higher-temperature, lower-density, softer foamed TPU, while darker regions correspond to lower-temperature, higher-density, harder TPU.

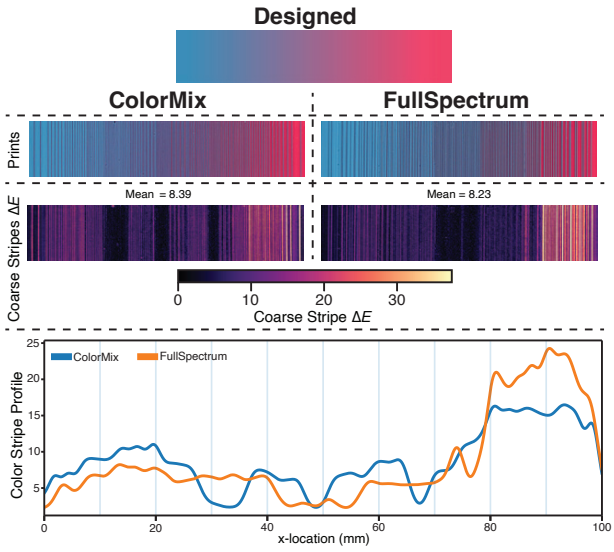


Figure 15: Material-fraction gradient reproduced through downstream halftoning workflows. The source design varies linearly from cyan to magenta and is exported to both Prusa ColorMix and Orca FullSpectrum using the same CMYKW PLA filaments. Coarse-banding analysis shows similar mean banding over the full sample, and both methods produce stronger low-frequency artifacts near the gradient endpoints where one filament has low participation.

as cyan, magenta, yellow, black, and white (CMYKW). The compiler does not perform the final filament-level halftoning. Instead, it exports ColorMix- and FullSpectrum-compatible `.3mf` projects containing the virtual recipes and sub-mesh assignments that PrusaSlicer or OrcaSlicer consume during slicing.

4.3.1 Material-Fraction Gradients. We first evaluate direct material-fraction compilation using a 100 mm × 20 mm × 20 mm rectangular bar. The source design defines a linear two-material volume-fraction field from 100% cyan to 100% magenta along the x-axis. We set the partition count to $N = 20$, and the compiler discretizes the field into 20 mixture regions before exporting the same abstract material-fraction design to both Prusa ColorMix and Orca FullSpectrum. Both samples are printed on the same Prusa XL with the same CMYKW PLA filaments, a 0.4 mm nozzle, and 0.2 mm layers.

Figure 15 compares the two printed gradients and analyzes the spatial structure of their halftoning artifacts. This example evaluates how each downstream slicer realizes a prescribed material-fraction field. We therefore focus on the spatial structure of the printed halftone pattern, measuring coarse color banding along the gradient rather than treating the rendered cyan–magenta interpolation as a calibrated color target. Each photograph is converted to CIELAB space, and a local estimate of the intended cyan–magenta gradient is subtracted so that the remaining residual emphasizes halftoning artifacts. We then compute a sliding-window spectral analysis along the 100 mm sample length. Within each window, the metric integrates residual color variation at long spatial periods orthogonal to the gradient direction. Higher values indicate wider, more visible halftone bands; lower values indicate finer alternation that visually blends more smoothly.

ColorMix and FullSpectrum produce similar mean coarse-banding scores over the full bar, with means of 8.39 and 8.23, respectively. Additionally, the spatial profile shows that both methods produce more visible low-frequency banding near the ends of the gradient, where one material participates at a low fraction. This artifact becomes especially apparent below 25 mm and above 75 mm, corresponding to mixtures dominated by one filament. The high-magenta end shows stronger visible striping, which is consistent with the observed perceptual dominance of magenta over cyan in these prints. Although the mean scores are similar, FullSpectrum shows a larger coarse-banding peak above 75 mm than ColorMix. We therefore use this experiment to constrain the full-color palette in the next section: nonzero components in mixed-filament recipes must be at least 25%.

4.3.2 Full-Color Reproduction. We next evaluate full-color compilation from an sRGB attribute field. The source design is a mountain-scene image, shown in Figure 16, imported as an RGB color field and extruded volumetrically into a 150 mm × 110 mm × 10 mm object. The same source design is compiled to Prusa ColorMix and Orca FullSpectrum using the same CMYKW PLA base filaments. For both workflows, the palette-selection algorithm receives the same manufacturer-reported color values for the loaded CMYKW base filaments, listed in Appendix A.2.

The compiler selects a bounded palette from printable recipes containing either a single physical filament or mixtures of two or three base filaments. Mixed recipes must satisfy the 25% minimum nonzero component fraction motivated by the material-fraction

gradient result in Fig. 15. ColorMix and FullSpectrum use different recipe prediction models, so the optimizer may select different recipes even though both workflows start from the same physical CMYKW filament colors. Appendix A.2 reports the selected palettes for the mountain example.

Figure 16 compares the printed mountain samples. We photograph each sample under fixed camera settings, lighting, and alignment, register the photograph to the digital reference, and compare the images in CIELAB space using CIEDE2000 color difference, ΔE_{00}. Lower ΔE_{00} indicates closer color reproduction. Before computing ΔE_{00}, we Gaussian filter both the reference and printed sample images in CIELAB space with $\sigma = 12$ pixels. This filtering emphasizes regional color reproduction rather than bead-scale or pixel-scale variation.

We also compute an excess texture metric to isolate fabrication artifacts. First, we remove the smooth color component from both the printed image and the reference image, leaving only fine-scale residual variation. We then compare these residuals and subtract the residual variation already present in the reference. Higher values therefore indicate additional high-frequency texture introduced by the printing process.

ColorMix reproduces the mountain benchmark more accurately than FullSpectrum under these conditions. The ColorMix sample has a mean ΔE_{00} of 11.73, compared with 16.63 for FullSpectrum. ColorMix also produces a smoother printed image, with mean excess texture of 3.04 compared with 6.69 for FullSpectrum. The error and texture maps in Fig. 16 show that FullSpectrum introduces stronger horizontal banding across large regions of the image, while ColorMix preserves smoother sky and mountain regions. These results show that, for this CMYKW PLA palette and benchmark image, the compiler can target both slicer workflows from the same volumetric color design, and the downstream workflow choice produces measurable differences in color fidelity and texture quality.

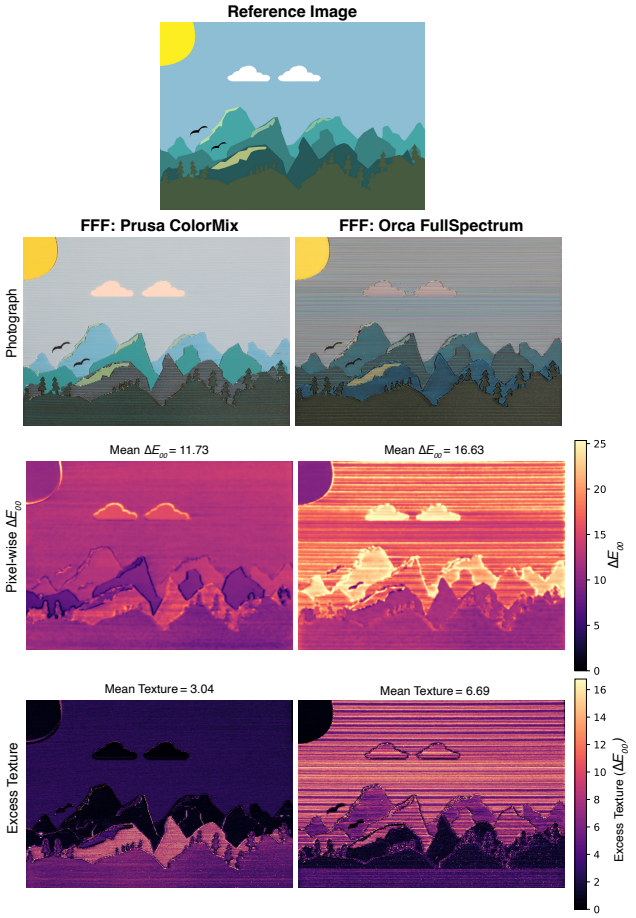


Figure 16: Full-color reproduction from one volumetric RGB color design compiled to two downstream halftoning workflows. Photographs are registered to the reference image and compared using Gaussian-filtered CIEDE2000 color difference, ΔE_{00}, to measure regional color fidelity. Excess texture isolates high-frequency artifacts introduced by fabrication. For this CMYKW PLA benchmark, ColorMix produces lower mean color error and lower excess texture than FullSpectrum.

4.4 Reduction in Manual Slicer Interactions

The preceding examples use the slicer interface as a compilation target rather than as a manual authoring environment. We quantify this difference by counting the number of repetitive slicer interactions that a user would need to recreate the same assignments manually. We define one interaction as a single click or one grouped text-entry action. The counts assume PrusaSlicer 2.9.6 and a favorable manual baseline: the model is already partitioned into the required sub-meshes, all sub-meshes are already aligned, and the user only needs to apply the correct settings or material assignments. The count therefore excludes mesh generation, file export, import, alignment, repair, naming, and slicing. It measures only the repetitive assignment work that the compiler replaces.

Table 3 lists the primitive interaction counts used in this estimate. Setting infill density requires four interactions per sub-mesh, setting both fuzzy-skin parameters requires nine interactions per sub-mesh, and assigning a material or logical tool requires three interactions per sub-mesh. Our compiler writes these assignments directly into the `.3mf` project, so the corresponding manual assignment count is zero for the compiled workflow.

Table 4 applies these primitive counts to the examples in this paper. For joint partitions, the reported totals assume the worst case in which every cell of the Cartesian-product partition is nonempty and corresponds to a unique settings combination, such that each region produces a separate sub-mesh. The infill-density bar in Fig. 6a contains 12 sub-meshes, each requiring an infill-density assignment, for a total of 48 manual interactions. The fuzzy-skin

Table 3: Primitive manual interaction counts used to estimate repetitive slicer assignment effort in PrusaSlicer 2.9.6.

ID	Manual assignment task	Interactions
1	Set both fuzzy-skin parameters	9
2	Set infill density	4
3	Assign material or logical tool	3

Table 4: Estimated manual slicer interactions required to recreate the assignments used in the paper examples. The compiled workflow embeds these assignments directly in the generated `.3mf` project.

Example	Meshes	Task IDs	Interactions
Infill bar	12	2	48
Infill bracket	6	2	24
Fuzzy sheet, 5 × 5	25	1	225
Fuzzy sheet, 10 × 10	100	1	900
Fuzzy bunny	144	1	1,296
Foaming fuzzy bunny	216	1, 3	2,592
Volume-fraction gradient	20	3	60
Mountains	10	3	30

bunny uses a 12 × 12 joint partition over two fuzzy-skin parameters, yielding at most 144 sub-meshes and 1,296 manual interactions. The foaming fuzzy bunny combines fuzzy-skin settings with virtual-extrusion tool assignments over a 6×6×6 joint partition. Under the same worst-case assumption, recreating this project manually would require both fuzzy-skin assignment and logical-tool assignment for each of 216 sub-meshes, for a total of 2,592 interactions.

These counts show the scaling problem that motivates slicer project compilation. Manual assignment effort grows linearly with the number of generated regions and with the number of independent controls assigned to each region. This estimate further assumes that the user has already completed the non-trivial tasks of partitioning the model and maintaining exact alignment among the resulting regions. A manual workflow may also introduce unnecessary or inconsistent partitions, whereas the compiler constructs the partition directly from the user-specified discretization parameters. By embedding settings, material assignments, and virtual-tool assignments during compilation, the proposed workflow removes this repetitive interface work while preserving the downstream slicer's native planning and preview capabilities.

4.5 Runtime Benchmarking

We report runtime benchmarking to evaluate whether slicer project compilation remains practical as partition count increases. We use a 200 mm × 50 mm × 50 mm rectangular prism and compile three single-gradient benchmarks. The settings-mesh benchmark applies a grid infill-density gradient from 5% to 95% along the bar. The virtual-extrusion benchmark applies a temperature gradient from 200°C to 250°C, with flow-rate compensation computed from Eq. 1. The half-toning benchmark applies a broad sRGB color gradient and exports color-recipe assignments. For each case, we vary the number of actual exported partitions, record compiler wall-clock time, and slice the generated `.3mf` project in PrusaSlicer 2.9.6 for a Prusa XL.

Figure 17a shows that compiler time increases approximately linearly for all three targets. Half-toning has the largest constant factor because it serializes color-recipe metadata, while settings meshes and virtual extrusion remain lower-cost. At 40 partitions, the largest compile time is about 60 s, and the other two targets remain near 20 s. Downstream slicing also remains tractable (Fig. 17b). Virtual extrusion and half-toning scale close to linearly, while grid infill-density settings meshes grow faster, likely because many local infill regions increase toolpath-planning complexity in PrusaSlicer. Even so, the longest slicing time is approximately 21 s.

Project size follows the same trend (Fig. 17c). Settings meshes and virtual extrusion have nearly identical linear storage costs, while half-toning produces larger files because each partition carries color-recipe assignments. The largest project remains below 25 MB. These results show that increasing partition count, and therefore gradient fidelity, does not produce prohibitive compile times, slicing times, or project sizes over the tested range.

5 Conclusion

We present slicer project compilation as an automated bridge between heterogeneous attribute fields and slicer-native fabrication workflows. Rather than requiring a designer to manually partition a graded object, export aligned meshes, assign region settings, configure material recipes, or post-process G-code, the compiler emits configured `.3mf` projects that conventional slicers can load, preview, slice, and fabricate. The method reduces continuous attributes to labeled sub-mesh regions and serializes those regions as local slicer settings, process-state assignments, or filament-mixture recipes. Across the examples, the same compilation framework drove slicer-planned infill and fuzzy-skin variation, virtual-extrusion control of nozzle temperature and flow rate, and color or material-fraction assignments for downstream half-toning workflows.

The results also demonstrate that slicer project compilation becomes more useful when paired with calibrated translation models. For temperature responsive foaming filaments, we characterized the relationships among nozzle temperature, flow-rate compensation, density, and Shore A hardness for TPU and PLA workflows. These models allow a designer to specify high-level property fields, such as density or Shore hardness, while the compiler receives the lower-level temperature and flow-rate fields required for fabrication. The density-shifted balance beam, Shore-hardness gradient, and foaming fuzzy bunny show how these translations can be used within a single automated workflow for heterogeneous FFF fabrication. In this workflow, high-level design intent, calibrated material-process behavior, and slicer-ready project generation remain separate steps, but they operate together without manual slicer assignment.

Future work could focus on addressing the following limitations. Continuous fields are approximated by a finite number of project regions, so higher spatial fidelity increases project size and downstream slicer workload. The calibrated process models depend on the material, printer, and print profile, and therefore must be remeasured when the fabrication setup changes. The generated projects also depend on slicer-specific `.3mf` dialects and on the process-control mechanisms exposed by each slicer. These constraints motivate extensions to additional slicer backends and, more broadly, slicers that can natively accept and plan over true heterogeneous gradients, including continuous toolpath-aware transitions.

By compiling heterogeneous attributes into slicer-ready projects, the system makes field-based design usable within existing FFF workflows while preserving the path planning, preview, support generation, and printer-profile infrastructure of mature slicers. The

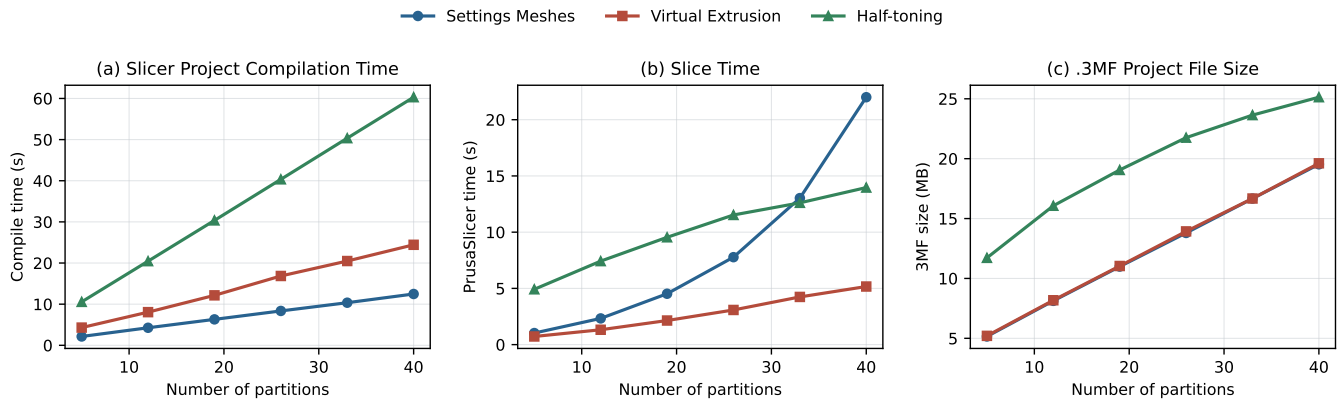


Figure 17: Runtime and storage scaling for slicer project compilation. A 200 mm × 50 mm × 50 mm **prism is compiled with increasing numbers of exported partitions for grid infill-density settings meshes, temperature-driven virtual extrusion with flow-rate compensation, and color-recipe half-toning. (a) Compiler time grows approximately linearly. (b) PrusaSlicer 2.9.6 slicing time remains below** 21 s**, although grid infill-density settings meshes grow fastest. (c) Project size increases approximately linearly and remains below** 25 MB**.**

open-source release provides a reusable implementation compatible with the OpenVCAD design representation, including project-generation backends and calibrated foaming-filament translation models. We intend this release to support further work by the community on functionally graded design, material-process characterization, and automated fabrication workflows in which high-level spatial intent can be carried through to printable artifacts without manual slicer reconstruction.

Acknowledgments

This material is based upon work supported by the Charles Stark Draper Laboratory, Inc. under Contract No. N00030-24-C-6001. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Strategic Systems Programs.

A Appendix

A.1 Code Availability

The implementation, example projects, calibration data, and scripts used to produce the figures are available at: https://github.com/MacCurdyLab/OpenVCAD-Public

A.2 Color Palette Recipes

This appendix reports the filament colors and selected virtual recipes used for the full-color mountain benchmark in Sec. 4.3.2. Table 5 lists the physical CMYKW PLA filaments loaded in the printer. Table 6 lists the bounded palettes selected for the ColorMix and FullSpectrum exports. Both workflows use the same physical filament set, but their recipe prediction models differ, so the selected virtual mixtures are not identical.

Table 5: Base CMYKW PLA filaments loaded for both ColorMix and FullSpectrum export. The colors supplied to the palette-selection algorithm are the manufacturer-reported hex values for the physical filaments.

Slot	Color	Hex	Description
E1		#0082AD	cyan
E2		#D10F4F	magenta
E3		#EFD00B	yellow
E4		#3D3E3D	near black
E5		#E6EAEF	white

References

Vahid Babaei, Kiril Vidimče, Michael Foshey, Alexandre Kaspar, Piotr Didyk, and Wojciech Matusik. 2017. Color contoning for 3D printing. *ACM Transactions on Graphics* 36, 4 (Aug. 2017), 1–15. doi:10.1145/3072959.3073605

Michael Borish, Brian T. Gibson, Cameron Adkins, and Paritosh Mhatre. 2022. Automated Process Planning for Embossing and Functionally Grading Materials via Site-Specific Control in Large-Format Metal-Based Additive Manufacturing. *Materials* 15, 12 (June 2022), 4152. doi:10.3390/ma15124152

A. R. Damanpack, André Sousa, and M. Bodaghi. 2021. Porous PLAs with Controllable Density by FDM 3D Printing and Chemical Foaming Agent. *Micromachines* 12, 8 (July 2021), 866. doi:10.3390/mi12080866

Bradley Duncan, Robert D. Weeks, Benjamin Barclay, Devon Beck, Patrick Bluem, Roberto Rojas, Maxwell Plaut, John Russo, Sebastien G. M. Uzel, Jennifer A. Lewis, and Theodore Fedynyshyn. 2023. Low-Loss Graded Dielectrics via Active Mixing of Nanocomposite Inks during 3D Printing. *Advanced Materials Technologies* 8, 3 (Feb. 2023), 2201496. doi:10.1002/admt.202201496

Anthony Garland and Georges Fadel. 2015. Design and Manufacturing Functionally Gradient Material Objects With an Off the Shelf Three-Dimensional Printer: Challenges and Solutions. *Journal of Mechanical Design* 137, 11 (Nov. 2015), 111407. doi:10.1115/1.4031097

Brian T. Gibson, Bradley S. Richardson, Tayler W. Sundermann, and Lonnie J. Love. 2019. Beyond the Toolpath: Site-Specific Melt Pool Size Control Enables Printing of Extra-Toolpath Geometry in Laser Wire-Based Directed Energy Deposition. *Applied Sciences* 9, 20 (Oct. 2019), 4355. doi:10.3390/app9204355

Joshua T Green, Ian A Rybak, Jonathan J Slager, Mauricio Lopez, Zachary Chanoi, Calvin M Stewart, and Roger V Gonzalez. 2023. Local composition control using an active-mixing hotend in fused filament fabrication. *Additive Manufacturing Letters* (2023), 100177. doi:10.1016/j.addlet.2023.100177

Zachary C. Kennedy and Josef F. Christ. 2020. Printing polymer blends through in situ active mixing during fused filament fabrication. *Additive Manufacturing* 36 (Dec. 2020), 101233. doi:10.1016/j.addma.2020.101233

Tim Kuipers, Eugeni Doubrovski, and Jouke Verlinden. 2017. 3D hatching: linear halftoning for dual extrusion fused deposition modeling. In *Proceedings of the 1st*

Table 6: Selected palette entries for the mountains example. Swatches show the selected target color; recipes list the physical-filament fractions used to reproduce that color. ColorMix and FullSpectrum use the same physical CMYKW PLA base filaments but different recipe prediction models, so the selected palettes differ.

ColorMix			FullSpectrum		
ID	Target	Recipe	ID	Target	Recipe
6	#8DBED3	E5 75%, E1 25%	6	#8DBED3	E5 75%, E1 25%
7	#45593F	E4 50%, E1 25%, E3 25%	7	#45593F	E1 50%, E2 25%, E3 25%
8	#E2E64E	E3 50%, E5 50%	8	#397F7F	E4 50%, E1 25%, E5 25%
9	#397F7F	E1 75%, E3 25%	9	#FFF200	E3 75%, E5 25%
5	#FDFEFE	E5 100%	10	#255757	E4 75%, E1 25%
10	#255757	E4 50%, E1 25%, E5 25%	11	#B1BF7F	E5 50%, E3 25%, E4 25%
11	#47A6A6	E1 50%, E5 50%	5	#FFFFFF	E5 100%
12	#A1C89D	E5 50%, E1 25%, E3 25%	12	#BBDBA5	E5 50%, E1 25%, E3 25%
13	#FFF200	E3 75%, E5 25%	13	#47A6A6	E1 50%, E5 50%
4	#010101	E4 100%	14	#91BE9B	E1 50%, E3 25%, E5 25%

Annual ACM Symposium on Computational Fabrication. ACM, Cambridge Massachusetts, 1–7. doi:10.1145/3083157.3083163

Tim Kuipers, Willemijn Elkhuizen, Jouke Verlinden, and Eugeni Doubrovski. 2018. Hatching for 3D prints: Line-based halftoning for dual extrusion fused deposition modeling. *Computers & Graphics* 74 (Aug. 2018), 23–32. doi:10.1016/j.cag.2018.04.006

Francesco Leoni, Pierandrea Dal Fabbro, Stefano Rosso, Luca Grigolato, Roberto Meneghello, Gianmaria Concheri, and Gianpaolo Savio. 2023. Functionally Graded Additive Manufacturing: Bridging the Gap between Design and Material Extrusion. *Applied Sciences* 13, 3 (Jan. 2023), 1467. doi:10.3390/app13031467

Yan Li, Zuying Feng, Liang Hao, Lijing Huang, Chenxing Xin, Yushen Wang, Emiliano Bilotti, Khamis Essa, Han Zhang, Zheng Li, Feifei Yan, and Ton Peijs. 2020. A Review on Functionally Graded Materials and Structures via Additive Manufacturing: From Multi-Scale Design to Versatile Functional Properties. *Advanced Materials Technologies* 5, 6 (June 2020), 1900981. doi:10.1002/admt.201900981

Guang Liu, Wuzhen Huang, Yaohui Wang, Huilin Ren, Guoquan Zhang, Limin Zhou, and Yi Xiong. 2024. Stress field-aware infill toolpath generation for additive manufacturing of continuous fiber reinforced polymer composites. *Materials & Design* 239 (March 2024), 112756. doi:10.1016/j.matdes.2024.112756

Giselle Hsiang Loh, Eujin Pei, David Harrison, and Mario D. Monzón. 2018. An overview of functionally graded additive manufacturing. *Additive Manufacturing* 23 (Oct. 2018), 34–44. doi:10.1016/j.addma.2018.06.023

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics* 21, 4 (Aug. 1987), 163–169. doi:10.1145/37402.37422

Thomas J. Ober, Daniele Foresti, and Jennifer A. Lewis. 2015. Active mixing of complex fluids at the microscale. *Proceedings of the National Academy of Sciences* 112, 40 (Oct. 2015), 12293–12298. doi:10.1073/pnas.1509224112

Jason M. Ortega, Melody Golobic, John D. Sain, Jeremy M. Lenhardt, Amanda S. Wu, Scott E. Fisher, Lemuel X. Perez Perez, Adam W. Jaycox, James E. Smay, Eric B. Duoss, and Thomas S. Wilson. 2019. Active Mixing of Disparate Inks for Multi-material 3D Printing. *Advanced Materials Technologies* 4, 7 (July 2019), 1800717. doi:10.1002/admt.201800717

Mehmet Ozdemir and Zjenja Doubrovski. 2023. Xpandables: Single-filament Multi-property 3D Printing by Programmable Foaming. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM, Hamburg Germany, 1–7. doi:10.1145/3544549.3585731

Joshua S. Pelz, Nicholas Ku, William T. Shoulders, Marc A. Meyers, and Lionel R. Vargas-Gonzalez. 2021. Multi-material additive manufacturing of functionally graded carbide ceramics via active, in-line mixing. *Additive Manufacturing* 37 (Jan. 2021), 101647. doi:10.1016/j.addma.2020.101647

Prusa Research. 2026. Print with Dozens of Colors: Our New Open-Source ColorMix for EasyPrint and PrusaSlicer. Accessed: 2026-06-16.

Prusa Research. n.d.a. Fuzzy Skin. Prusa Knowledge Base. Accessed: 2026-06-16.

Prusa Research. n.d.b. Per Model Settings. Prusa Knowledge Base. Accessed: 2026-06-16.

ratdoux. 2026. OrcaSlicer-FullSpectrum: G-code Generator for Snapmaker U1 with Full Spectrum Layer Blending. GitHub repository. Accessed: 2026-06-16.

Tim Reiner, Nathan Carr, Radomír Měch, Ondřej Šťava, Carsten Dachsbacher, and Gavin Miller. 2014. Dual-color mixing for fused deposition modeling printers. *Computer Graphics Forum* 33, 2 (May 2014), 479–486. doi:10.1111/cgf.12319

Eder Sales, Tsz-Ho Kwok, and Yong Chen. 2021. Function-aware slicing using principal stress line for toolpath planning in additive manufacturing. *Journal of Manufacturing Processes* 64 (April 2021), 1420–1433. doi:10.1016/j.jmapro.2021.02.050

Haichuan Song, Jonàs Martínez, Pierre Bedell, Noémie Vennin, and Sylvain Lefebvre. 2019. Colored Fused Filament Fabrication. *ACM Transactions on Graphics* 38, 5 (Oct. 2019), 1–11. doi:10.1145/3183793

Daniele Tammaro, Massimiliano Maria Villone, and Pier Luca Maffettone. 2022. Microfoamed Strands by 3D Foam Printing. *Polymers* 14, 15 (Aug. 2022), 3214. doi:10.3390/polym14153214

Silvia Vesco and Daniel Salvi. 2025. Fuzzy skin in fused filament fabrication: enhancing morphology, wettability, and friction through a full-factorial experimental plan. *Progress in Additive Manufacturing* 10, 12 (Dec. 2025), 11233–11257. doi:10.1007/s40964-025-01285-0

Kiril Vidimče, Szu-Po Wang, Jonathan Ragan-Kelley, and Wojciech Matusik. 2013. OpenFab: a programmable pipeline for multi-material fabrication. *ACM Transactions on Graphics* 32, 4 (July 2013), 1–12. doi:10.1145/2461912.2461993

Charles Wade, Devon Beck, and Robert MacCurdy. 2025a. Implicit Modeling for 3D-printed Multi-material Computational Object Design via Python. In *Proceedings of the ACM Symposium on Computational Fabrication*. ACM, Cambridge MA USA, 1–16. doi:10.1145/3745778.3766659

Charles Wade, Devon Beck, and Robert MacCurdy. 2025b. Implicit toolpath generation for functionally graded additive manufacturing via gradient-informed slicing. *Additive Manufacturing* 112 (Aug. 2025), 104963. doi:10.1016/j.addma.2025.104963

Charles Wade, Devon Beck, and Robert MacCurdy. 2026. Design-Intent Compilation for Heterogeneous Fabrication. arXiv:2607.22741 [cs.GR] https://arxiv.org/abs/2607.22741

Charles Wade, Graham Williams, Sean Connelly, Braden Kopec, and Robert MacCurdy. 2024. OpenVCAD: An open source volumetric multi-material geometry compiler. *Additive Manufacturing* 79 (2024), 103912. doi:10.1016/j.addma.2023.103912

Lingwei Xia, Sen Lin, and Guowei Ma. 2020. Stress-based tool-path planning methodology for fused filament fabrication. *Additive Manufacturing* 32 (March 2020), 101020. doi:10.1016/j.addma.2019.101020

Jiangping Yuan, Guangxue Chen, Hua Li, Hartmut Prautzsch, and Kaida Xiao. 2021. Accurate and Computational: A review of color reproduction in Full-color 3D printing. *Materials & Design* 209 (Nov. 2021), 109943. doi:10.1016/j.matdes.2021.109943

Wei Yuan, Xiaoli Zhao, Shujun Li, and Yue Zhu. 2022. Effect of laser scanning speed on microstructure and mechanical properties of SLM porous Ti-5Al-5V-5Mo-3Cr-1Fe alloy. *Frontiers in Materials* 9 (Aug. 2022), 973829. doi:10.3389/fmats.2022.973829