Secure 3D Printing: Reconstructing and Validating Solid Geometries using Toolpath Reverse Engineering

Nektarios Georgios Tsoutsos
New York University
nektarios.tsoutsos@nyu.edu

Homer Gamil
New York University Abu Dhabi
og532@nyu.edu

Michail Maniatakos
New York University Abu Dhabi
michail.maniatakos@nyu.edu

ABSTRACT

As 3D printing becomes more ubiquitous, traditional centralized process chains are transformed to a distributed manufacturing model, where each step of the process can be outsourced to different parties. Despite the countless benefits of this revolutionary technology, outsourcing parts of the process to potentially untrusted parties raises security concerns, as malicious design modifications can impact the structural integrity of the manufactured 3D geometries. To address this problem, we introduce a novel compiler that allows reverse engineering G-code toolpaths (i.e., machine commands describing how a geometry is printed) to reconstruct a close approximation of the original 3D object. Our framework then uses Finite Element Analysis to simulate the reconstructed object under different stress conditions and validate its structural integrity, without requiring a golden model reference.

Keywords

Additive Manufacturing; 3D Object Reconstruction; Finite Element Analysis; Constructive Solid Geometry

1. INTRODUCTION

The premise of 3D printing technology lies in its inherent potential to build artifacts with arbitrary internal features, rapidly and with relatively low cost [8]. Otherwise known as Additive Manufacturing (AM), 3D printing overcomes the fundamental limitations of traditional (subtractive) methods, such as milling or turning, where material is selectively removed from solid blocks and the final geometry is subject to tool accessibility. Indeed, it is impossible to feature fully enclosed hollow spaces (e.g., a ship inside a bottle) using subtractive methods as tool access is blocked [7]. At the same time, 3D printing machines operate by depositing materials in thin layers, which eventually allows constructing any complex geometry with significantly less material waste. Due to its numerous advantages, this disruptive technology is also considered by many as the next industrial revolution [2, 9].

At the same time, there is no shortage of concerns on the security implications of 3D printing, especially when one leverages the decentralized nature of contemporary process chains and outsources manufacturing steps to potentially untrusted third parties [17]. In this threat model, it is important to ensure the structural integrity of the 3D printed geometries against intentional and unintentional defects. Specifically, our goal is to be able to detect any modification that has an adverse impact to the structural integrity of the printed artifact, especially when its load bearing capacity is weakened. As a motivational example, one may consider the original design of a solid rod that is modified to feature an internal (hidden) void; such defect could decrease the rod's ability to bear loads and cause a premature failure.

In practice, it is possible to simulate 3D geometries under load or stress conditions using numerical methods, such as Finite Element Analysis (FEA) [3], which can be implemented using solvers for systems of partial differential equations (e.g., SfePy [4]). Nevertheless, FEA is a method generally applicable on pure 3D geometries (i.e., the source files, which is the output of the 3D design software), while 3D printing machines operate over sequences of thin layers (also called "slices") represented in a special numerical control language called "G-code" [11]. The G-code is created through compilation of the pure 3D geometry, which is a *lossy process*, and is not meaningful for FEA. Moreover, to preserve their Intellectual Property (IP) and discourage unauthorized modifications to the original source, designers usually convert the source files into stereolithography (STL) format [15], which is not applicable for FEA either, as STL files encode only the surface geometry of the polyhedral approximation of the original 3D object.

In this work in progress, we introduce a novel reconstruction and validation framework that allows reverse-engineering any compiled G-code file to a close approximation of the original 3D geometry source. Our methodology addresses the fundamental problem of detecting structural integrity defects given compiled/printer-level representations of an object, as the reverse-engineered 3D source is naturally compatible with FEA simulations, while G-code or STL versions are not. In a globalized 3D printing process chain, the entity receiving an IP design in compiled file format (e.g., as G-code) can painlessly validate its structural integrity through simulation, while avoiding the printing time overhead or the material cost required for validating a physically printed object. In effect, our approach acts as "anti-malware" for 3D

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CPSS'17, April 02 2017, Abu Dhabi, United Arab Emirates
© 2017 ACM. ISBN 978-1-4503-4956-7/17/04...$15.00
DOI: http://dx.doi.org/10.1145/3055186.3055198

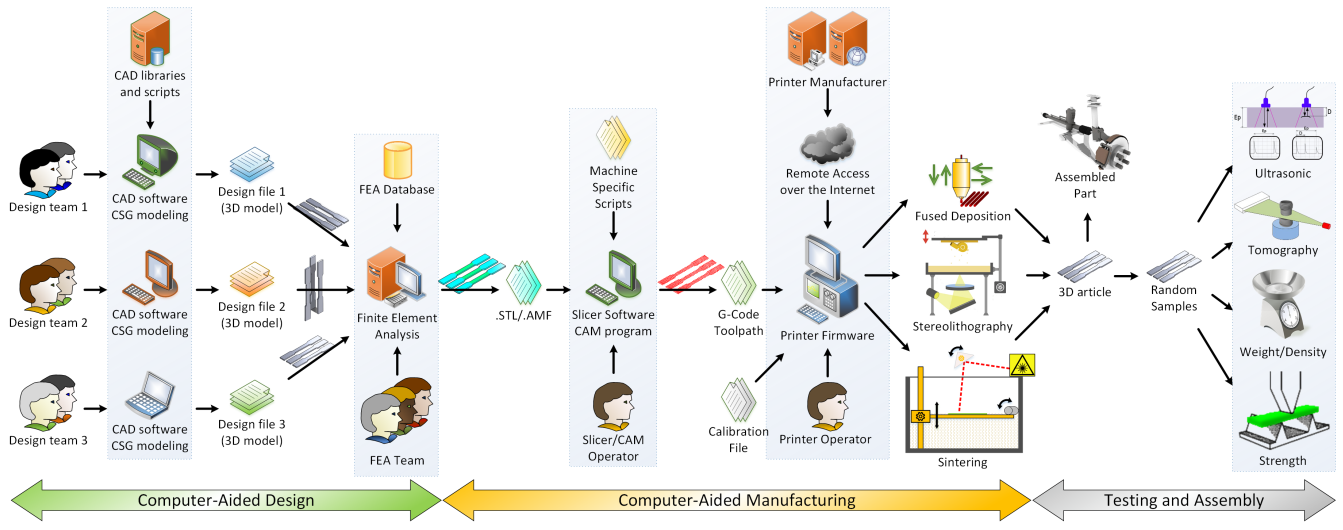

Figure 1: A typical distributed process chain for modern 3D printing.

objects, without the need of a golden model, as FEA simulations can verify if stress specifications are met.

The centerpiece of our contribution is GSim: a new compiler that converts G-code files to constructive solid geometry (CSG) source code [12]. Our compiled CSG output is a pure 3D geometry that is naturally compatible with FEA and matches the granularity (i.e., the size of distinct features) of the target 3D printer. This implies that any details lost compared to the original 3D source are primarily attributed to the lossy conversion of the original IP to G-code (or polyhedral approximation in STL) before these inputs are ever used by GSim. Moreover, our framework seamlessly integrates GSim with the OpenSCAD CSG compiler [10], the Gmsh 3D mesh generation tool [6] and the SfePy FEA simulator [4], allowing streamlined structural validation of untrusted IP designs.

The rest of the paper is organized as follows: in Section 2 we discuss background notions in AM, including typical distributed 3D printing process chains, while in Section 3 we elaborate on our threat model and potential attack scenarios. In Section 4, we present the details of our reconstruction and validation framework, as well as the benefits and limitations of our GSim compiler, and in Section 5 we present our evaluation case study. The paper concludes with our final remarks in Section 6.

2. 3D PRINTING BACKGROUND

2.1 Distributed Process Chain Description

As illustrated in Fig. 1, a typical process chain for distributed 3D printing consists of three major steps [17]:

- The Computer-Aided Design (CAD) step, where different design teams use 3D design software/libraries and CSG modeling to build alternative 3D geometries satisfying specific constraints for a given problem.[1] To validate the alternative designs and select the best fit for the target application, a FEA team simulates each 3D geometry under different loads using parameters from material databases (e.g., [1]). As FEA simulations can visually highlight weak or vulnerable regions (e.g., using yellow or red color), a golden model comparison is typically not required.

- The Computer-Aided Manufacturing (CAM) step, where the best candidate design of the previous step is exported as a polyhedral approximation (STL file) and send to "slicer" software. The slicer converts the polyhedral surface information into thin parallel layers, and generates a "toolpath" for traversing each such layer so that its entire surface can be covered by the printer head. The slicer software ultimately stores the path information as a sequence of printer-specific instructions in the form of G-code, and the toolpath is forwarded to the 3D printer. The printer itself consists of a dedicated computer running special firmware that drives mechanical components (i.e., actuators and motors), while the printer sensors are calibrated to ensure dimensional accuracy.[2]

- The Testing step, where the physical properties of the printed artifact are assessed either destructively (using a representative sample) or non-destructively. Non-destructive tests may include (a) *ultrasonic testing* that can identify sizable impurities and manufacturing problems over a single axis; (b) *tomography methods* such as X-ray and CT scanning that project radiation through the 3D artifact (over different angles) to visualize its internal structure in high resolution and detect impurities; (c) *weight/density tests* that act as *physical hash digests* and can be used as probabilistic indicators of manufacturing problems.[3] Conversely, *destructive strength tests* involve destroying a sample by applying force in a consistent manner, to identify if the printed object matches the expected strength requirements.

[1]Assuming independent CAD teams, radically different 3D designs are not uncommon across teams.

[2]Network connectivity may also be available to enable remote management/assistance from the printer manufacturer, as well as to obtain licenses and firmware updates.

[3]Mass and volume measurements are more effective against non-malicious modifications as they are not collision-resistant indicators.

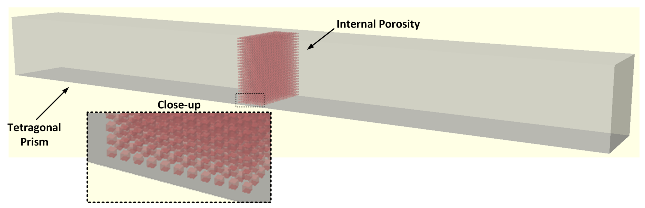

Figure 2: Porosity is maliciously added to a 3D printed tetragonal prism. This attack introduces minute voids in very close proximity to each other (e.g., in a crystal structure) and depending on the pore size and distance, the affected area is expected to fail prematurely under stress. In sintering printers, disabling the laser signal to introduce porosity would let unmelted material remain within the generated void, so the overall density and weight of the object remain unchanged.

2.2 3D Printing Methods

During the CAM step of the process, alternative printing methods may be used. One popular option is Fused Deposition Modeling (FDM), where a thin plastic filament is melted by a hot print nozzle and ejected/deposited according to the toolpath generated by the slicer software. Another popular method is Sintering, where raw material powder (usually metal or nylon) is spread by a moving roller over a fusion bed and particles are melted on the spot using a fine heat source (e.g., a laser beam). Likewise, Stereolithography methods use photosensitive resins and focused ultraviolet laser beams to selectively polymerize the liquid raisin within a container and form each layer of the 3D artifact.

3. THREAT MODEL AND ATTACKS

3.1 Design and Manufacturing Attacks

From a security perspective, the distributed nature of the 3D printing process chain, as elaborated in Section 2.1, makes it vulnerable to different attacks that may affect the structural integrity of the printed artifact. During the CAD step of the process, where the original IP is created, a malicious insider may corrupt the 3D model or introduce structural defects directly, so that the physical object would fail prematurely or under lower stress levels. The adversary may launch the attack by exploiting the underlying CAD software or the used 3D model database and scripts. Alternatively, the adversary may attack the FEA software to report false design problems (e.g., nominate the least suitable design etc.) or poison the FEA database to falsify material properties (e.g., indicate that the 3D object is metal while it is actually plastic). Performing FEA simulations at a later step in the process would help detecting such threats.

Moving forward to the CAM step of the process, where the original 3D source is compiled into polyhedral abstractions (i.e., an STL file), we are concerned about threat actors modifying the surface information (e.g., removing polyhedrons) so that voids can be introduced at critical points of the structure. Similarly, by adding malicious protrusions or making dimension transformations/scale changes, the adversary may directly impact the elasticity and load-bearing capacity of the object. When the STL file is sliced into toolpath information (i.e., G-code), maliciously modified slicer software may be used to degrade specific printing parameters, such as the skin/core porosity, the head speed and acceleration, the nozzle and print chamber (curing) temperature, as well as the extrusion speed. Likewise, an adversary may corrupt the firmware of the 3D printer itself to modify how G-code commands are interpreted by the printer actuators, falsify sensor values or inject additional commands to the print stack. Such attacks may introduce the following effects:

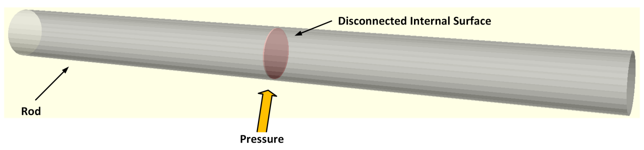

Figure 3: A 3D printed rod with a disconnected internal plane. This attack creates a void in the core of the rod, leaving only the external "skin" intact; as a result, the mechanical integrity of the rod under perpendicular stress is weakened, and since the volume of the void is very small, the relative change in weight/density measurements is minimal.

- unwanted internal porosity (Fig. 2), disconnected internal planes (Fig. 3) or unsintered material, by selectively disabling the printing head or altering the skin/core exposure depths;
- reduced structural integrity, by altering the toolpath orientation (e.g., reversing X-Y coordinates [17]), increasing the print head speed, skipping layers, or reducing the curing temperature;
- unexpected disintegration (applicable to multi-head printers), by replacing the base print material with support material, intended to temporarily hold the printed layers in place, in the presence of gravity;[4]
- physical damage to the 3D printer or the printed object itself by exceeding the X-Y path limits, or forcing the nozzle onto the object or the print bed.

3.2 Attacks During Testing

The 3D printing process is also vulnerable during the testing step, where an adversary may try to tamper with the test results. In tomography or ultrasonic scan tests, a potential threat may be the reduction of scan resolution, which effectively hides minute defects (such as porosity) introduced in the CAD or CAM steps. Likewise, an adversary may falsify the test results directly (causing defective objects to pass the tests), or tamper with the testing parameters to decrease the actual yield rates and cause legitimate 3D objects to be discarded as faulty. In the former case, accepting faulty objects may cause indirect financial impact due to expensive recalls afterwards, while the latter case causes direct financial impact due to legitimate objects being discarded. Similarly, in case "physical hash digests" are used (such as weight/density tests), attackers could easily falsify the expected measurements by finding weight/volume collisions between the original and the modified object, masquerading faulty artifacts as legitimate. For example, by displacing material from a critical region to another area, the weight/density measurements would remain the same while the strength of the object could be weakened.

[4]Typically, support material is dissolved by submerging the multi-material object into an oxidizer bath (e.g., acid), so it is possible to carve unexpected voids if support material is used where base material is expected.

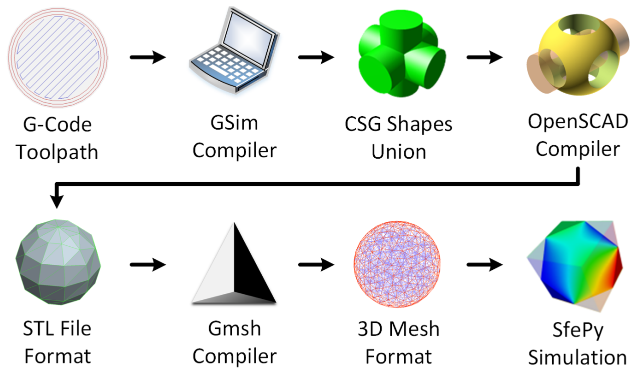

Figure 4: Steps of our reconstruction methodology.

4. SECURE 3D PRINTING USING RECONSTRUCTION AND VALIDATION

As discussed in the previous sections, the distributed nature of modern 3D printing process chains and the uncertainty of outsourced manufacturing to potentially untrusted parties, introduces security risks to the structural integrity of the manufactured object. In this work, our goal is to detect such attacks as late in the process as possible, yet before (or concurrently as) the object is actually manufactured, to reduce our attack exposure only to the printing step itself, and avoid the cost associated with manufacturing/testing any defective objects. For that matter, we focus on reverse-engineering G-code command sequences to reconstruct a valid *pre-image* that would closely approximate the given G-code. The latter is then transformed to a 3D mesh and simulated under stress conditions using FEA.[5]

Our reconstruction methodology is summarized to the following steps (Fig. 4):

- Our developed GSim compiler parses each G-code statement and generates a corresponding set of CSG instructions. Essentially, for each G-code statement, GSim identifies the material added to the current layer (in 3D space) and outputs the CSG union of basic solid shapes (e.g., prisms or cylinders) at the corresponding orientation and coordinates (Fig. 5 and 6). These solid shapes can also be envisioned as interconnected filament sections of a virtual FDM printer, floating in the three dimensional space based on the X, Y and Z coordinates of the G-code toolpath. A judicious choice of this virtual 3D filament shape (e.g. prism, cylinder, etc.), as well as its profile diameter, renders any 3D surface and solid.

- The CSG code is rendered to a solid 3D object using a compatible solid geometry compiler (in this work we use OpenSCAD [10]), and then saved as an STL file. This reconstructed STL file describes the same polyhedral approximation as the original STL file (i.e., the one generated by the original designer).

- The reconstructed STL file is then converted to a 3D mesh (in this work we use Gmsh [6]), which essentially fills the closed surface represented by the STL input. The generated 3D mesh is now compatible with FEA tools (without loss of generality, in this paper we use the SfePy FEA toolchain [4]).

- The 3D mesh is simulated under different stress conditions (e.g., linear elasticity test [4]) and the simulation result is returned as a color coded image illustrating any overstressed areas. In case such areas are identified under nominal stress conditions, the G-code provided in the first step is problematic, as it would manufacture an object with inadequate stress tolerance. Conversely, if the FEA simulation does not highlight any problematic areas, this provides assurance that the G-code to be printed is problem-free. Notably, our FEA simulations do not require a golden model, since only the material parameters and force specifications are required to simulate an object under stress conditions.

[5]Note that the FEA simulation of the reconstructed 3D object is separate from the FEA simulation performed by the designers during the CAD step.

```
module c_arc(x_cnt, y_cnt, rad, d_end, c_sz, dir)
{
  d_step = 2 * asin(c_sz / (2 * (rad + c_sz / 2)));
  for (d = [360 * dir : d_step * (1 - 2 * dir) : d_end])
  {
    translate([x_cnt + rad * cos(d),
               y_cnt + rad * sin(d), 0])
    rotate([0, 0, d])
    cube(c_sz, center=true);
  }
  translate([x_cnt + rad * cos(d_end),
             y_cnt + rad * sin(d_end), 0])
  rotate([0, 0, d_end])
  cube(c_sz, center=true);
}
```

Figure 5: Code snippet of a CSG function implementing a virtual filament arc defined by "G2" and "G3" G-code operations. GSim interprets 3D printer head movements and filament extrusion commands from each G-code operation, before returning the corresponding CSG instructions to be compiled by OpenSCAD [10]. The baseline version of GSim supports 20 different RepRap-compatible G-code operations [16], as shown in Table 1.

```
module c_line(x0, y0, z0, x, y, z, c_sz)
{
  hull()
  {
    translate([x0, y0, z0])
      cube(c_sz, center=true);
    translate([x, y, z])
      cube(c_sz, center=true);
  }
}
```

Figure 6: Code snippet of a CSG function implementing a straight virtual filament line defined by "G0" and "G1" G-code operations.

The benefits of our novel reconstruction and validation methodology can be summarized as follows:

- **Cost:** the proposed reconstruction method incurs negligible costs, compared to printing an actual 3D object and testing it for structural integrity defects. If the provided G-code is malicious, the defects would be detected without wasting any "printer time" or material.

- **Speed:** our proposed implementation simulates G-code commands and compiles the CSG instructions with negligible time overhead (within a few seconds). Rendering 3D objects and STL files from CSG is done using existing tools (i.e., OpenSCAD), and the corresponding overhead is in the order of minutes. This is

Table 1: Basic RepRap-compatible G-code operations recognized by GSim [16].

Operation(s)	Description
`G0`, `G1`	Linear Movement
`G2`, `G3`	Arc Movement
`G20`, `G21`	Inches/Millimeters
`G28`	Return to Origin
`G90`, `G91`	Absolute/Relative Position
`G92`	Reset Current Position
`M73`	Report Build Percentage
`M82`, `M83`	Absolute/Relative Extruder Position
`M84`	Disable Motors
`M101`, `M103`	Enable/Disable Extruder
`M104`, `M109`	Set Extruder Temperature
`M106`, `M107`	Enable/Disable the Fan

significantly faster compared to printing an actual artifact, where the time overhead can be in the order of hours.

- **Effectiveness:** as demonstrated in our case study evaluation (Section 5), the proposed reconstruction framework provides visual identification of structural defects in G-code that manifest as voids. GSim can also simulate reduced temperature settings that cause insufficient material curing, which is represented in our reconstructions as loosely coupled three dimensional filament sections with small inter-filament gaps.

- **Uniqueness:** to the best of the authors' knowledge, GSim is the first compiler that allows reversing the slicing step of the 3D printing process chain and convert G-code statements back to STL format.

In addition to the benefits discussed in the previous paragraph, our reconstruction and validation approach also has some inherent limitations, based on our threat model (Section 3). First, since our reconstruction is based on the G-code input to the 3D printer, any attack that happens within the printer itself (e.g., G-code statement injection, printer firmware modification, etc.) cannot be detected, as its effect appears after the G-code would be validated. Moreover, our approach may be limited with respect to detection granularity; specifically, depending on the slicing layer height during the original slicing step (i.e., the original G-code generation), as well as the size of the smallest features supported by the target 3D printer, the reconstructed STL files (used for FEA simulation) are polyhedral approximations of the original 3D object and STL file. We note, however, that this limitation is not inherent to the GSim compiler itself, rather a side effect of the granularity of the provided G-code commands. Hence, if the proposed method is used after coarse-granularity slicing of the original STL, the reconstruction itself would not be identical to the original 3D object, since slicing is an inherently lossy transformation and information is already lost before GSim is engaged.[6]

5. CASE STUDY EVALUATION

To evaluate our reconstruction and validation methodology, we used a tetrahedral prism as our case study. Due to its geometric simplicity and orthogonal dimensions, a tetrahedral prism offers several benefits over alternative geometries (such as a cylinder or a sphere for example), since it allows precise alignment of the prism's edges with the generated G-code toolpath. The latter effectively minimizes the information lost during the lossy slicing step (i.e., before our framework is engaged), which is beneficial for evaluation purposes to illustrate the accuracy of our reconstruction method itself.

To generate the G-code toolpath for a tetrahedral prism object, which will be used as input to our framework, we created an original 3D source using CSG and OpenSCAD. This 3D source corresponds to the CAD step in Fig. 1, and instead of OpenSCAD, any 3D design software (e.g., [1, 5]) could have been used without loss of generality. It is also important to note that this use of CSG and OpenSCAD should not be confused with our reconstruction framework that leverages the same tools for a different purpose. To generate RepRap-compatible G-code, the 3D design was first translated into STL file format by OpenSCAD and then sliced using Slic3r software [13]. This G-code would normally be send to a 3D printer to manufacture the 3D object, but in our case, it is sent directly to our framework for reconstruction and validation.

The reconstruction result for the non-defective tetrahedral prism is presented in Fig. 7, where no impurities are identified as expected. Following the steps described in Section 4 and summarized in Fig. 4, the input G-code was compiled to CSG using GSim, before being compiled to STL by OpenSCAD. The STL file was converted to 3D mesh format using Gmsh and then simulated by SfePy. For our FEA, we used *linear elasticity* from [4], which simulated a displacement of 0.01 units applied only on the positive half of the X axis (while the negative half remained fixed), assuming a linear elastic material with $\lambda = 10$ and $\mu = 1$ Lamé constants [14]. Knowing the exact material used by the 3D printer, alternative Lamé constants can be used in the simulation. Likewise, depending on the strength tolerance specifications of the object, different displacement values at different regions of the object can be applied as well.

To compare against the non-defective SfePy simulation output, we repeated the previous experiment using a defective tetrahedral prism. The latter was also constructed in OpenSCAD using CSG, but this time small voids were introduced into the prism geometry using subtractive CSG commands. As before, the defective 3D source was converted to STL and then sliced to a G-code toolpath. In realistic scenarios, this defective/malicious G-code is what our framework is designed to detect, acting as anti-malware for the 3D printer.

After invoking our reconstruction and validation framework on the G-code of the defective prism, we obtained a new SfePy simulation, which demonstrates multiple defects (Fig. 8). Specifically, on the right end of the prism, the simulation indicates multiple foci of stress, which indicates the existence of voids and impurities. At the same time, a less prominent impurity is visible across the long axis of the

[6]Even with finer-granularity slicing, the reconstructed STL is an *image* of the same 3D object, but the byte contents of the STL files would generally look different; hence, file comparisons (e.g., `diff`) would be ineffective for assessing equivalence of 3D objects.

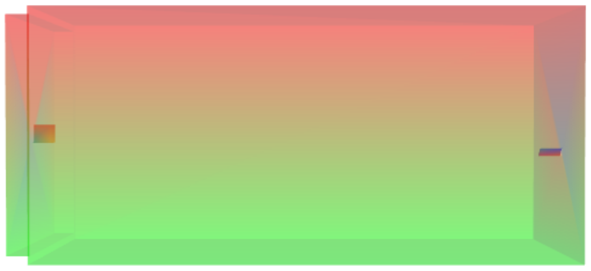

Figure 7: SfePy simulation output for a non-defective tetrahedral prism, reconstructed using our methodology (Fig. 4). As expected, the simulation does not show any abnormal stress points.

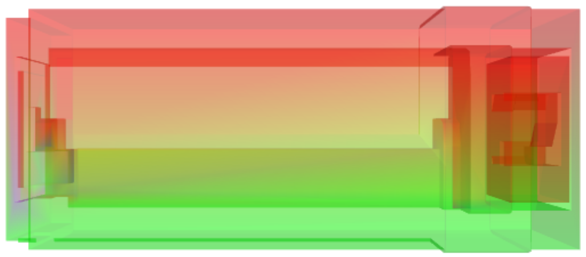

Figure 8: SfePy simulation of a defective tetrahedral prism, reconstructed using our methodology (Fig. 4). Compared to Fig. 7, the simulation clearly shows sizable stress points and impurities.

prism. In addition, the unexpected stress to the prism under displacement conditions on the positive half of the X axis is illustrated with the more intense, non-gradual, red colored areas. At the same time, the negative half of the X axis, which is not displaced in the simulation, remains (mostly) green.

Our case study evaluation shows the effectiveness of our methodology in identifying defects in G-code toolpaths of 3D objects, without the use of a golden model. Using SfePy simulations along with knowledge of the stress tolerance specifications and material parameters of the 3D printed object, we can validate if the input G-code meets our specific requirements without having to actually print and test a physical artifact.

6. CONCLUDING REMARKS

In this work in progress, we have developed a complete framework for reconstructing 3D objects by reverse engineering G-code toolpaths. Our reconstructions are then used in FEA simulations to validate the structural integrity of the 3D object corresponding to the original G-code, without the need to actually print or test the 3D geometry. A key part of our contribution is GSim, a novel compiler that converts numerical control toolpath commands into CSG instructions that enable reconstructing a 3D object. In future work, we will broaden compatibility of the framework to additional G-code variants (in addition to RepRap) supporting more 3D printer types, while improving the accuracy of FEA simulations through automated assignment of material stiffness parameters and displacement values.

Acknowledgments

This work was partially supported by the NYU Abu Dhabi Global Ph.D. Student Fellowship program.

7. REFERENCES

[1] Autodesk. Inventor: Mechanical design and 3D CAD software. [Online] http://www.autodesk.com/products/inventor/.

[2] B. Berman. 3-D printing: The new industrial revolution. *Business horizons*, 55(2):155–162, 2012.

[3] J. Chaskalovic. *Finite element methods for engineering sciences: theoretical approach and problem solving techniques*. Springer Science & Business Media, 2008.

[4] R. Cimrman. SfePy - write your own FE application. In P. de Buyl and N. Varoquaux, editors, *Proceedings of the 6th European Conference on Python in Science (EuroSciPy 2013)*, pages 65–70, 2014. Resources: http://sfepy.org.

[5] Dassault Systèmes. Solidworks: 3D CAD design software. [Online] http://www.solidworks.com/.

[6] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[7] I. Gibson, D. W. Rosen, B. Stucker, et al. *Additive manufacturing technologies*, volume 238. Springer, 2010.

[8] N. Guo and M. C. Leu. Additive manufacturing: technology, applications and research needs. *Frontiers of Mechanical Engineering*, 8(3):215–243, 2013.

[9] S. H. Huang, P. Liu, A. Mokasdar, and L. Hou. Additive manufacturing and its societal impact: a literature review. *The International Journal of Advanced Manufacturing Technology*, 67(5-8):1191–1203, 2013.

[10] M. Kintel. The Programmers Solid 3D CAD Modeller. [Online] http://www.openscad.org/.

[11] T. R. Kramer, F. M. Proctor, and E. Messina. The NIST RS274/NGC Interpreter-Version 3. *National Institute of Standards and Technology (NIST), NISTIR*, 6556, 2000.

[12] D. H. Laidlaw, W. B. Trumbore, and J. F. Hughes. Constructive solid geometry for polyhedral objects. In *ACM SIGGRAPH computer graphics*, volume 20, pages 161–170. ACM, 1986.

[13] A. Ranellucci. Slic3r: G-code generator for 3d printers. [Online] http://slic3r.org/.

[14] J. Salençon. *Handbook of Continuum Mechanics: General Concepts - Thermoelasticity*. Springer Science & Business Media, 2012.

[15] M. Szilvśi-Nagy and G. Matyasi. Analysis of STL files. *Mathematical and Computer Modelling*, 38(7):945–960, 2003.

[16] The RepRap Project. RepRap Firmware G-codes. [Online] http://reprap.org/wiki/G-code.

[17] S. E. Zeltmann, N. Gupta, N. G. Tsoutsos, M. Maniatakos, J. Rajendran, and R. Karri. Manufacturing and security challenges in 3D printing. *JOM*, pages 1–10, 2016.