Beyond Layers: A 3D-Aware Toolpath Algorithm for Fused Filament Fabrication

Samuel Lensgraf[†] and Ramgopal R. Mettu[†,*]

***Abstract*— Fused filament fabrication (FFF) is gaining traction for rapid prototyping and custom fabrication. Existing toolpath generation methods for FFF printers take as input a three-dimensional model of the target object and construct a layered toolpath that will fabricate the object in 2D slices of a chosen thickness. While this approach is computationally straightforward, it can produce toolpaths that can contain significant, yet unnecessary, extrusionless travel.**

In this paper we propose a novel 3D toolpath generation paradigm that leverages local feature independence in the target object. In contrast to existing FFF slicing methods which print an object layer by layer, our algorithm provides a means to print local features of an object without being constrained to a single layer. The key benefit of our approach is a tremendous reduction in "extrusionless travel," in which the printer must move between features without performing any extrusion. We show on a benchmark of 409 objects that our method can yield substantial savings in extrusionless travel, 34% on average, that can directly translate to a reduction in total manufacturing time.

I. Introduction

The rise of additive manufacturing has given users unprecedented freedom and flexibility in manufacturing objects of arbitrary geometry. In general, the part is created by a computational determination of a tool path which is executed by the machine. *Fused filament fabrication*[1] (FFF), a widely used form of additive manufacturing builds a model incrementally by depositing substrate that becomes fused to other, already extruded substrate. Fused filament fabrication allows more flexibility in choice of material than other methods of additive manufacturing but sacrifices speed in doing so. A major limitation of this approach is the large amount of machining time (hours to days) that is required to print complex parts. When printing models with multiple distinct parts, the printer typically spends a large fraction of its time traveling between extrusions. The standard tool path generation process exacerbates the problem since the printer must complete all instructions on a given layer before moving on to the next.

We propose an algorithm to minimize the wasted motion of an FFF tool path. As input, our algorithm takes a three-dimensional model (e.g., in STL format) and produces a tool path in which the printer can move in all three dimensions so as to reduce extrusionless travel. Our algorithm utilizes the same toolpath segmentation as existing methods (i.e., layers consisting of paths), but utilizes a notion of local dependencies between parts of the model to ensure that printing and movement in three dimensions produces the target model and is collision-free. To our knowledge, this is the *first* such approach to be applied to FFF toolpath generation. We believe our approach presents a new approach to toolpath generation that can optimally leverage tool geometry to reduce fabrication time. Our algorithm is simple. In testing it requires an order of magnitude less time than slicing, and works with compact geometric approximations of the print head and carriage. Over a large benchmark of 409 models, we demonstrate an average of 34% reduction in the wasted motion of slicing-based toolpaths.

A. Related Work

Tool path generation for 3D printers derives from the computer numerical control (CNC) milling problem. Current path planning algorithms for 3D printers utilize the adaptive and nonadaptive iso-planar tool path generation techniques. Adaptive iso-planar tool path generation changes the layer height used to describe a model according to the surface complexity of the part thereby reducing total manufacturing time [7]. Various techniques for deciding layer height based on a geometric characterization of the part called adaptive slicing have been proposed [19], [20]. For a comprehensive review of slicing techniques we refer the reader to a review by Pandey et al. [16].

The (nonadaptive) iso-planar technique takes a model from a CAD application and slices it with parallel planes yielding a series of contours. These contours can then be approximated by a series of contact points which the print head must interpolate. Han et al. provide a useful characterization of speeding the additive manufacturing

[†]Department of Computer Science, Tulane University, New Orleans, LA 70118

∗ Corresponding author: rmettu@tulane.edu.

[1]We note that the this type of printing is also commonly referred to as *fused deposition modeling* (FDM), but since the term is trademarked by Stratasys, we use untrademarked term *fused filament fabrication.*

process [9]. Several techniques have been proposed to apply optimization procedures to tool paths to reduce the total "airtime" (i.e. extrusionless travel) needed to produce a part. Wah et al. provide a technique to limit the total tool path length within layers of a general layered manufacturing problem [23]. Castelino et al. [4] provide a traveling salesperson with precedence constraints characterization of CNC machining, and show good results in minimizing the airtime required to manufacture a part. Wojcik et al. provide a genetic algorithm to minimize the tool path length for 3D printers [24] by combining raster tool path segments together. Volpato et al. [22] provide two methods for reducing the total path length of a 3D printed part by combining printed regions according to optimization procedures. Each of these is similar to ours in that it attempts to reduce the extrusionless travel distance, but optimization is limited to considering the tool path in a single layer.

Other methods expedite the process by reducing the required amount of support volume [2]. Support structures are added in the tool path generation process to keep the output model from deviating from the original mesh. In many models, this support structure can drastically increase print time. Our algorithm assumes a constant support structure and could be used effectively with support structure reducing techniques. Jin et al. [10] provide a method for generating the infill structure that reduces the overall path length. Methods altering the infill structure to increase manufacturing efficiency are an important step in improving part strength and manufacturing time. Our approach does not change the input infill structure, and could be used efficiently in concert with methods like this. A method that decomposes a large input part into more easily manufacturable, thinner walled parts then packs them together to minimize the total volume of the part [21] has been recently developed. Again, this is an approach which would be effective in concert with ours.

Finally, methods called freeform 3D printing which use specialized hardware and materials to break the layer constraint [15] have seen success in many applications including large scale construction work [12]. Our methodology can be easily adapted to work with these methods as well.

II. Our Approach

Since traditional FFF printing proceeds layer-by-layer, it is easy to see that the target object can always be printed correctly (subject to support and printer constraints). Since our approach diverges from traditional methods, we first define our notion of printability. A path is *printable* if all parts of the path have a proper support structure and if the print head will not intersect already printed parts of the model while traversing the path [18]. Our method proceeds by first computing toolpath segments in each layer as in traditional methods, and then combining these into a 3D-aware toolpath that attempts to minimize extrusionless travel. We achieve this by abstracting the tool path and imposing a set of geometric constraints based on a notion of local dependence (and thus, printability).

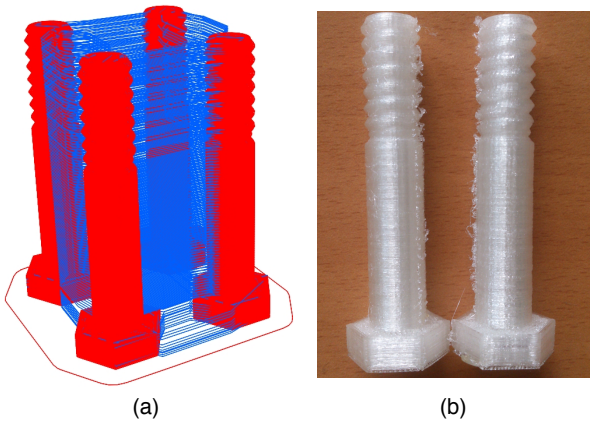

Fig. 1. (a) An example of a tool path with a large amount of wasted motion (blue) that could be avoided by printing the columns as independent objects rather than layer by layer. (b) Two of the screws produced when this path is printed.

For a simple motivating example of our approach, consider the tool path made for four identical screws shown in Figure 1. This is a pathological example for layer-by-layer printing, since each layer incurs wasted motion, causing the extrusionless travel to scale as the volume of the bounding box of the model. For the model in Figure 1, extrusionless travel accounts for 55375.6mm of the total 317030.0mm of motion required to print the model, or about 18% of the total motion. This accounts for a significant portion of both the time and energy required during printing. We propose a method that can, with an appropriate printer geometry, significantly improve upon such examples. A favorable printer geometry is one which allows us to meaningfully abstract the print head with a small bounding box. Any print head that hangs below the extruder can be abstracted. For example, on the Prusai3 printer we used to print the models in Figures 1 and 3 for testing, we can create a $7mm \times 7mm \times 7mm$ bounding box around the nozzle that hangs just below the rest of the extruder. In our testing, the Prusa i3 extruder geometry admits substantial savings over a large benchmark of models from popular open source mesh websites. In the next

Algorithm 1 Imports and optimizes a tool path F

Input: A tool path F, a specified tolerance ϵ, the radius of the extruder bounding box r, the height of the extruder bounding box h
Output: An optimized toolpath.

```
1: procedure OPTIMIZE(F, ϵ, r, h)
2:     segments ← PARSE(F)
3:     segments ← REMOVEEXTRUSIONLESS(segments)
4:     layers ← CREATELAYERS(segments)
5:     islands ← ∅
6:     for i = 0, ..., len(layers) do
7:         islands[i] ← CREATEISLANDS(layers[i], ϵ)
8:     end for
9:     for k = 0, ..., len(islands[0]) do
10:        islands[0][k].D ← ∅
11:    end for
12:    for i = 1, ..., len(islands) do
13:        L ← islands[i − 1, ..., i − n]
14:        for k = 0, ..., len(islands[i]) do
15:            islands[i][k].D ←
       CALCULATEDEPENDENCIES(islands[i][k], L, r)
16:        end for
17:    end for
18:    for i = 0, ..., len(islands) do
19:        for j = 0, ..., len(islands[i]) do
20:            TOCONTINUOUSPATH(islands[i][j])
21:        end for
22:    end for
23:    return BUFFEREDGREEDY(islands, h)
24: end procedure
```

section we discuss the structure of our algorithm in detail.

III. Algorithm Description

Since the generation of tool path segments in each layer of the input model is straightforward, for simplicity we describe our algorithm by assuming we are given a set of motion segments (i.e., G-code instructions) and the associated layer of each segment. In our current implementation, we make use of the open-source Slic3r [17] path generator, which follows the iso-planar method described in Section I-A. In the sections that follow we describe each phase of our algorithm. First, we parse the input segments (Section III-A). Next, the layers are divided into *islands*, or portions of the path that are convenient to print together in (Section III-B). Then, each island is assigned a set of dependencies that must be met before it can be put onto the final path (Section III-C). Finally, the islands are combined into a final path in Section III-D. Algorithm 1 (OPTIMIZE) gives a top-down overview of the complete algorithm. OPTIMIZE makes use of a number of subroutines; we provide detailed descriptions of these below.

A. Parsing the path and preparing the data

For clarity, we briefly describe how our algorithm represents the given set of motion segments in F. PARSE converts the input path to a sequence of *motion segments* in the order that they occur in the path by combining the contact points in F.

Definition 3.1: A **motion segment** (segment) l is a directed line segment specified by its three dimensional end points where $l.b$ marks the beginning of the motion and $l.e$ marks the end of the motion. A segment also stores a type variable *type* that is either *print* or *travel* where *print* refers to printed motion and *travel* refers to non-printed motion.

REMOVEEXTRUSIONLESS iterates through the given sequence and removes all segments with the type *travel*. CREATELAYERS aggregates the raw sequence of printed motions into a sequence of layers.

Definition 3.2: A **layer** L is a sequence of motion segments l_i with $l_i.b.z = l_i.e.z = l_j.b.z = l_j.e.z \ \forall l_i, l_j \in L$.
We assume the layers in the sequence *layers* in Algorithm 1 are in ascending order of their z values.

B. Aggregating layers into islands

Islands are sets of motions in a single layer which are contained by a closed outer path. Figure 2 shows an example of four islands in a layer. All parts that feature closed walls can be conveniently decomposed into islands. This concept is used in the path generation algorithms of both CuraEngine [6] and Slic3r [17]. In order to decompose a layer into islands, our algorithm first decomposes the layer into a sequence of *continuous* printed paths.

Definition 3.3: A sequence of motion segments $P = p_1, p_2, \ldots, p_n$ is **continuous** if for each $p_i, p_j \in P$ $\text{DIST}(p_i, p_j)$ is less than ϵ for $i = j - 1$, where $\text{DIST}(p_i, p_j)$ measures the Euclidean distance between the end point of p_i and the starting point of p_j. A continuous sequence of motion segments is referred to hereafter as a path.

A sequence of paths from the original sequence of motion segments in the layer is created via a linear scan, breaking the original sequence at places where the segments are far enough apart. Paths are labeled as either *closed* or *open* as follows.

Definition 3.4: A path $P = p_1, p_2, \ldots, p_n$ is labeled as **closed** if $\text{DIST}(p_n, p_1)$ is less than ϵ, otherwise P is said to be **open**.

As seen in Figure 2, an island can be understood as a closed outer path that contains several open or closed inner paths where the solid walls of the model form the

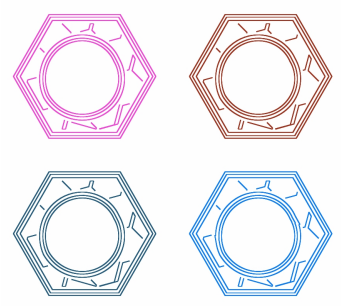

Fig. 2. An example of four islands in a set of machine instructions.

outermost path. Islands are a simplistic hierarchical sort of the type developed by Choi et al. [5]. The following definition provides more rigorous characterization.

Definition 3.5: An **island** I is a non-continuous set of paths with a single outer path and zero or more open or closed inner paths. The outer path contains all of the inner paths and is contained by no other path in the layer. Inner paths may contain other paths but must be contained by the outer path. Open paths may be the outer path of an island only in the case that they are contained by no other path.

A path p_1 contains another path p_2 if p_1 is closed and p_2 occurs entirely inside of p_1. That is, if all points of p_2 are inside of p_1. We assume that the paths do not cross, so if p_2 is contained in p_1, p_2 does not cross p_1. So, in order to test if p_1 contains p_2, we may simply verify whether $p_2[0].b$ is contained in p_1 via the winding number method [1]. The winding number method is a point in polygon algorithm that works in linear time for arbitrary polygons. This method operates by calculating how many times a polygon winds around a given point. If the polygon winds around the point a nonzero number of times, the point is contained in the polygon.

CREATEISLANDS operates by selecting a candidate outer path o for an island and scanning through the list of paths in the layer searching for paths that it contains and adding those incrementally to an island, removing them from the input list as they are encountered. If a path that contains o is found, that path becomes the candidate outer, and the iteration begins again. A special case where a layer contains no closed paths is handled by creating a single island from the layer. This occurs in "raft" layers created by the path generator.

In the worst case each path p_i is neither contained by nor contains another path. In such a case, the procedure must check each path remaining in the list for each iteration. If we assume that testing whether a path contains another takes at most some constant time C, this procedure runs in $O(n^2)$ time where n is the number of paths in the layer.

C. Calculating the dependencies of islands

Now that the layers have been divided into sequences of islands, the post-processor must impose physical constraints on the order in which the islands are printed. These constraints ensure the integrity of the output model. Specifically, they ensure that while the print head is following all of the paths in an island it will not collide with another island, and that when an island is printed, it will be resting on all islands that are a part of its support structure. It is assumed that the support structure of an island is adequate in the input model. These constraints are realized by storing a set of dependencies D in each island object that must be printed before the island in question can be printed.

The print head is modeled by an axis aligned rectangular volume that fully contains the region identified by the user as the *collison* region C. Filament is extruded from the middle of the bottom face of this rectangular volume. C has side length equal to $2r$ and height h. For example, on the Prusai3 fitted with a Greg's Extruder [8] used for our physical tests, this region has a radius of 7mm and a height of 7mm. For this setup, the collision region is defined by the nozzle.

Ideally, when calculating the dependencies of island i, C is swept along the outer path of each island j to generate j_c in the layers l_j preceding i within the height of C. i is then dependent on all $j_c \in l_j$ which it intersects. To make calculation simpler and speed implementation, an expanded axis aligned box which contains j_c is calculated and tested for intersection instead. This expanded box is obtained by moving the corners of the box a distance of r in the x and y axes.

Definition 3.6: An island i in a layer at height z_i is **dependent** on another island j at height z_j if the expanded axis aligned bounding box of j intersects the axis aligned bounding box of i and $z_i - h \leq z_j < z_i$.

CALCULATEDEPENDENCIES populates the dependency set of each island according to Definition 3.6. The inputs to this procedure are an island I which is to have its dependency list populated, a sequence of the islands in the layers before I, L, and the radius of the print head r. For each island i in L, the algorithm checks whether the bounding box of I intersects the bounding box of i.

CALCULATEDEPENDENCIES runs in time linear in the number of islands in L if we assume that the length of the outer path of each is bounded by some constant C.

Algorithm 2 Applies a greedy heuristic over a series of buffers to generate a new continuous path.

Input: A sequence of layers L, the height h of C.
Output: A continuous, ordered list of segments r.

```
1:  procedure BUFFEREDGREEDY(L, h)
2:      printed ← [0, 0, ..., 0]            ▷ len(printed) = #islands
3:      r ← ∅; bufmin ← L[0][0].z; Z ← 0
4:      islandBuf ← ∅
5:      for current = L[0] ... L[len(L) − 1] do
6:          Z ← current[0].z
7:          if |Z − bufmin| ≥ h then
8:              bufmin ← Z
9:              l ← GREEDY(islandBuf)
10:             r.add(SPAN(r[len(r) − 1], l[0]); r.add(l)
11:             islandBuf ← ∅
12:         end if
13:         if current is the last element in L then
14:             islandBuf.add(current)
15:             l ← GREEDY(islandBuf)
16:             r.add(SPAN(r[len(r) − 1], l[0]); r.add(l)
17:             break
18:         else
19:             islandBuf.add(current)
20:         end if
21:     end for
22:     return r
23: end procedure
```

D. Combining the islands into the final path

After dependencies are calculated, islands may be combined into a final continuous path containing both extrusionless travel and printed motions. First, the printed paths in each island must be connected into a single path by extrusionless travel. TOCONTINUOUSPATH converts an island to a continuous path by starting with the outer path and greedily connecting the rest of the paths in the island with extrusionless travel. After this procedure has been invoked, an island may be treated as a sequence of motion segments. These island-paths are then greedily combined into a single path using BUFFEREDGREEDY (Algorithm 2).

1) The buffered optimization approach: As we discussed previously, we must operate on the input model in a manner that accounts for the print head. That is, we consider chunks of height h, ensuring that the bounding box of the print head, C, is a meaningful abstraction. Above C there is no guarantee that parts of the printer do not fully sweep a layer while printing an island. Additionally, there may be parts of the printer outside of C that cannot be easily abstracted to simple geometric representation. For example, the nozzle of the print head extends about $7mm$ below the rest of the print head and carriage in the Prusai3 test printer with Greg's extruder, and can be easily bounded whereas the rest of the carriage and print head cannot be easily bounded.

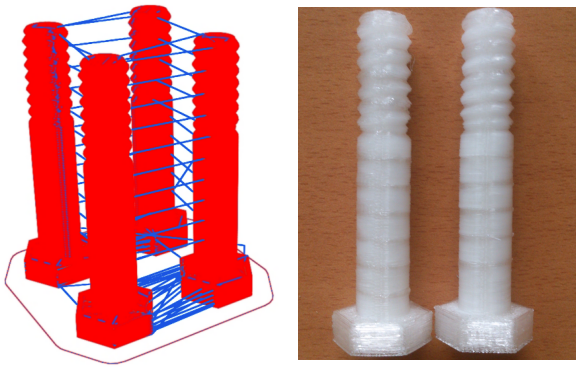

Fig. 3. (a) The path from Figure 1 after the optimization process. Blue lines are extrusionless travel. Red is printed motion. (b) Two of the screws that result from printing this path.

The input for BUFFEREDGREEDY is a sequence of sequences of island objects in which iterating over the outer sequence represents iterating over the layers of the model and the inner sequences are the islands in the appropriate layer. Here $L[0]$ is the first layer of the model, $L[0][0]$ is the first island in the first layer of the model, and $L[0][0][0]$ is the first motion segment in the first layer of the first island. Remember that the layers are ordered by z value, ascending. The outer portion of BUFFEREDGREEDY chunks the model into a series of smaller layer sequences and sends them through the inner GREEDY procedure. It is GREEDY that actually combines the islands into a single path and returns the path. So, BUFFEREDGREEDY divides the model into chunks and adds the connected paths from GREEDY onto its output list. GREEDY operates by connecting each island with its nearest possible neighbor. Each time an island is selected, it is marked as printed in the *printed* array. An island is possible if all of the islands in its dependency set are marked as printed. SPAN is used to connect islands together using extrusionless travel. It is discussed in the next subsection.

E. Ensuring collisionless inter-island travel

The SPAN procedure must ensure that lateral motion between islands does not cause collisions with already printed parts of the model. Ideally, we would compute the three-dimensional shortest collisionless path between the two islands in question. Unfortunately finding the shortest path point-to-point among polyhedral obstacles is in general NP-hard [3], and it is NP-complete even when restricted to axis-aligned boxes [14]. It is possible to efficiently compute shortest paths on the convex hull of the printed model, or to approximate true 3D shortest

paths. However these algorithms are not particularly fast, so to reduce the computation to reasonable levels in our implementation we simplify the task of routing between islands by converting it into a two dimensional problem. When routing between two islands, motions within the height of the current chunk purely in upward the z-axis are guaranteed not to cause collisions by the dependency constraints discussed earlier. In addition, islands at the current level of the print head cannot cause collisions with lateral motions because the extruder sits just above the level that it is currently printing at. We may simplify a 3D motion through a given chunk by moving the print head to the z-value of the higher of the two islands in question. All islands that are already printed above the higher island will be candidates for collision detection. We compress the set of bouding boxes of the already printed islands into two dimensions and then perform a union operation over all boxes in the plane. This yields a set of disjoint polygons suitable for constructing a visibility graph. We construct a visibility graph in this environment and query it to get our collisionless path. We note that this path is not necessarily the shortest possible path, since it might actually be beneficial for the print head to travel much further in the z direction for a reduced overall path.

In our current implementation, the construction of the visibility graph dominates the run time of our algorithm because a new visibility graph must be generated for each island. The construction of a visibility graph with a standard implementation takes $O(n^2 \log n)$ time where n is the number of vertices of the input polygons. Each island is approximated by a bounding box which has a constant number of vertices, so to run the SPAN procedure for each island costs $O(n^3 \log n)$ time where n is the number of islands. We discuss potential optimizations around this particular step in Section V.

This concludes the description of our algorithm. In the next Section, we present the results of our algorithm as compared to the Slic3r tool path generator for 3D printers.

IV. RESULTS

As a concrete example of our algorithm, we printed the model of four screws shown in Figure 1 after optimizing the path using our algorithm; the result is shown in Figure 3. With our method applied, we achieve a savings of about 30m of wasted motion. On our test printer, a RepRap Prusa i3, this translated to a total print time that was over an hour faster. As expected, printer geometry dictates the possible improvement; we can see in Figure 3 that printing in the z-direction is limited by h, the height of the bounding box of the print head.

For a more comprehensive evaluation, we constructed a benchmark set of 409 models of varying geometry taken from popular mesh sharing websites. Our benchmark set consists of models that have a wide variety of complexities in terms of the total distance traveled by the print head (65mm–9.6km), number of faces (36–2.9M), number of islands (2–35K) and the number of layers (2–2.7K). We provide a table of the full dataset along with individual complexity measures and various performance measurements for each model as a supplement to this manuscript.

To test such a large set of models, we made use of a simulator. The simulator was configured to read a tool path represented in the NIST RS274NGC G-code standard [11], compatible with the Marlin firmware [13]. The abstracted tool path could then be operated on and rendered back into a G-code representation and analyzed. We ran the simulator for each of the 409 models, using parameters appropriate for our test printer. As mentioned previously, the height of the bounding box of the print head was set to 7mm as was the radius.

To measure performance, we considered the overall reduction in extrusionless travel distance along with the overall time taken by our current implementation. Figure 4 provides a summary of the performance of our algorithm on our benchmark. The extrusionless travel in our models produced by a standard slicing-based toolpath (obtained with Slic3r) spans five orders of magnitude (Figure 4(a)), has a mean of 70m. Our primary figure of merit to characterize the overall improvement afforded by our algorithm is the percentage reduction of extrusionless travel. Figure 4(b) shows significant overall improvement on nearly all models. The mean and median percentage reductions in extrusionless travel was around 34% over all models. As is evident from the boxplot, our algorithm achieves over a 20% reduction in travel over three quarters of the benchmark. We did notice that our algorithm did not achieve significant reductions in a small fraction of cases. For example, about 30 models achieve an improvement of 5% or less. Importantly, we also saw an increase in the extrusionless travel in 20 models; 10 of these models had more than 5% greater extrusionless travel than before optimization. The worst two models increase extrusionless travel by 25% and 27% respectively. To understand the poor performance on these models, we examined the generated toolpaths and found two primary reasons for the increased extrusionles travel. First, when creating islands from motion segments, our algorithm does not optimize the placement of starting and endpoint points of the island. This is done by existing slicing methods, and

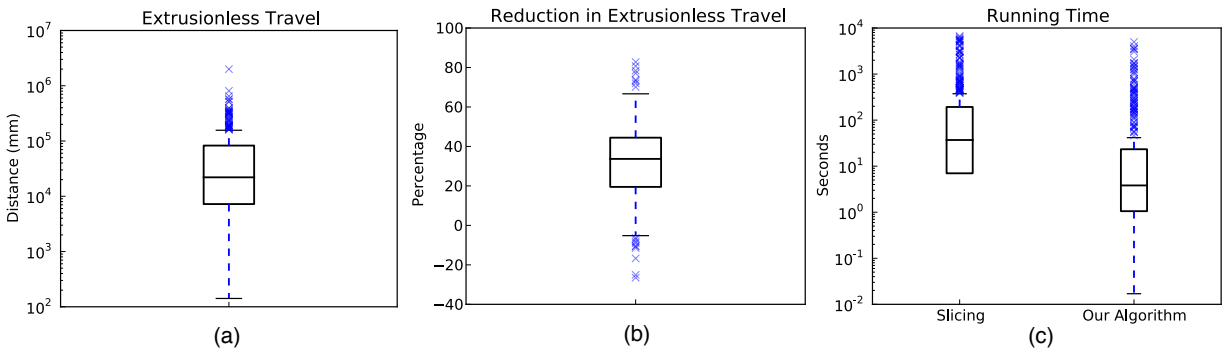

Fig. 4. **Performance.** Our approach provides a very low overhead means of significantly reducing extrusionless travel. The mean reduction in extrusionless travel over our benchmark of 409 models of varying complexity is about 34%. Moreover our optimization algorithm is computationally efficient, typically requiring taking just a few seconds.

usually makes use of a traveling salesperson heuristic. We did not implement a sophisticated heuristic for this step and thus incur a significant increase in extrusionless travel in some cases. We tested a simple simulated annealing approach to solve this issue and found it is able to achieve a positive reduction in travel in all but three of the worst-performing models. In the two models with largest increase in extrusionless travel, we observed a spatial organization of the model that causes poor performance. In examining the toolpaths created by these three models, we observed that our implementation successively chose a poor visitation order of islands on a given layer. This choice leads to a large penalty when there are many islands that are spatially distant (e.g., as in a ring of cylinders). This issue occurs infrequently and is addressable with further optimization.

To evaluate running time, we ran Slic3r and our implementation on a consumer-grade desktop for all models. The time taken by our algorithm is negligible in all but a few cases (see Figure 4(c)). Typically our algorithm takes on the order of seconds; the median runtime was 3.8 seconds. It is important to note that although our algorithm can be considered a standalone approach, our implementation takes the output of a slicing engine as input (so that islands do not have be computed). Thus the times given in Figure 4(c) are not meant to be head-to-head comparison. Instead we view the time reported as a reasonable upper bound on the additional overhead for our approach. In comparison to slicing, which has a median runtime of 38 seconds, the additional cost of our algorithm is on average just a few seconds. In cases with a relatively large running time, profiling showed that the construction of the visibility graph and computation of shortest path distances becomes costly in certain instances. That is, we must update the visibility graph and shortest paths after each greedy choice. This can be time consuming when the current topmost layer of islands is large. This aspect of the performance could be improved with a more efficient data structure and/or update scheme for the visibility graph.

A. *Discussion*

Overall, our approach provides a substantial reduction in extrusionless travel in the vast majority of models that we tested. It is important to note that since our method takes a fraction of the time required by slicing, cases in which extrusionless travel is actually increased can simply be eliminated by examining the output of our algorithm to judge whether there is an improvement. In cases where there is a minor improvement, issues such as part quality can also be considered in choosing a toolpath.

As shown empirically, our current algorithm can in some cases increase the amount of extrusionless travel. A natural question is to ask whether a 3D-aware method can achieve as much or less extrusionless travel than an optimal slicing-based method? This type of bound is an open question but it is possible to construct an illustrative example that shows that always printing independent elements is suboptimal. The high-level idea is that a 2D layer-by-layer approach can be ideal when it is possible to use an element of the model that divides two other elements as a mode of transportation between the two. If a 3D aware method is limited to printing spatially independent components in their entirety, then we can construct a worst-case input as follows. Let A and B be cubes with a small side length, and let D be a cube with very large side length positioned between

A and B such that it is touching neither A nor B. With this conformation, we may choose a print head radius so that A, B and D are mutually independent.

In the worst case, the starting and ending points of islands in D can be arranged to facilitate the traditional approach by allowing printed travel through D between A and B. In this case we need only use extrusionless travel to move from D to a smaller cube and to move up layers. We need only pay for the side length of D once – to move vertically up D between layers. With any ordering chosen by a 3D-aware method, the side length of D must be paid for twice – once to move up between layers, and once to move either between A and B or from D to A or B. We may choose the size of D relative to A and B to get arbitrarily bad outputs.

In practice, we find that realistically-sized examples are hard to put into conditions as dire as this pathological example due to the required proportions. Nevertheless this example provides some guidance as to how one may pursue a guaranteed reduction in extrusionless travel as compared to slicing. In particular, this example shows that independence between print segments is necessary but not sufficient. An additional notion of spatial separation is needed, and within appropriate regions must be able to balance the penalty of extrusionless travel with gains made in printing in three dimensions.

V. Conclusion

In this paper we have presented a novel, 3D-aware approach to toolpath planning for fused filament fabrication. Our algorithm makes use of printer geometry to model dependencies between local features in order to create a toolpath that attempts to minimize extrusionless travel. Existing slicing methods cannot take advantage of the possible savings in time, which we have shown is 34% on average. Our algorithm is relatively simple in its approach, but opens up a number of interesting questions about toolpath planning in three dimensions.

We are currently exploring improved methods for 3D aware planning. In particular we have taken inspiration from the cases in which our current method actually increases extrusionless travel. These cases demonstrate that while our greedy method works well in the vast majority of inputs, there is still improvement to be gained by improving the modeling of spatial separation in local features of the input model.

References

[1] D. G. Alciatore, "The point in polygon problem for arbitrary polygons," *Computational Geometry*, vol. 20, pp. 131–144, 2001.

[2] P. Alexander and D. Dutta, "Layered manufacturing of surfaces with open contours using localized wall thickening," *Computer-Aided Design*, vol. 32, pp. 175–189, 2000.

[3] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *Proceedings of the 28th IEEE Foundations of Computer Science*, 1987, pp. 29–60.

[4] K. Castelino, R. D'Souza, and P. K. Wright, "Tool-path optimization for minimizing airtime during machining," *Journal of Manufacturing Systems*, August 2004.

[5] S. Choi and K. Kwok, "A topological hierarchy-sorting algorithm for layered manufacturing," *Rapid Prototyping Journal*, vol. 10, pp. 98–113, 2004.

[6] "Cura engine source code repository," https://github.com/Ultimaker/CuraEngine.

[7] S. Ding, M. A. Mannan, and A. Poo, "Adaptive iso-planar tool path generation for machining of free-form surfaces," *Computer-Aided Design*, vol. 35, pp. 141–153, 2003.

[8] "Greg's hinged extruder," http://reprap.org/wiki/Gregs's_Hinged_Extruder.

[9] W. Han, M. A. Jafari, and K. Seyed, "Process speeding up via deposition planning in fused deposition-based layered manufacturing," *Rapid Prototyping Journal*, vol. 9, pp. 212–218, 2003.

[10] Y. Jin, Y. He, J. Fu, W. Gan, and Z. Lin, "Optimization of tool-path generation for material extrusion-based additive manufacturing technology," *Additive Manufacturing*, pp. 1–16, 2014.

[11] T. R. Kramer, F. M. Proctor, and E. R. Messina, "The nist rs274ngc interpreter - version 3," *NIST Interagency/Internal Report (NISTIR)*, August 2000.

[12] E. Krassenstein, "Branch technology 3d prints building walls with worlds largest freeform 3d printer launches 3d printed home competition," http://3dprint.com/85215/branch-3d-printed-walls/, July 2015.

[13] "Marlin firmware source repository," https://github.com/MarlinFirmware/Marlin.

[14] J. S. B. Mitchell and M. Sharir, "New results on shortest paths in three dimensions," in *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, ser. SCG '04. New York, NY, USA: ACM, 2004, pp. 124–133.

[15] N. Oxman, J. Laucks, M. Kayser, E. Tsai, and M. Firstenberg, "Freeform 3d printing: Towards a sustainable approach to additive manufacturing," *Green Design, Materials and Manufacturing*, 2013.

[16] P. M. Pandey, N. V. Reddy, and S. G. Dhande, "Slicing procedures in layered manufacturing: a review," *Rapid Prototyping Journal*, vol. 9, no. 5, pp. 274–288, 2003.

[17] "Slic3r gcode generator," http://www.slic3r.org, accessed: 4-29-2015.

[18] K. A. Tarabanis, "Path planning in the proteus rapid prototyping system," *Rapid Prototyping Journal*, vol. 7, pp. 241–252, 2001.

[19] K. Tata, G. Fadel, A. Bagchi, and N. Aziz, "Efficient sliciing for layered manufacturing," *Rapid Prototyping Journal*, vol. 4, pp. 151–167, 1998.

[20] J. Tyberg and J. Bohn, "Local adaptive slicing," *Computer-Aided Design*, vol. 28, pp. 307–318, 1998.

[21] J. Vanek, J. A. G. Galicia, B. Benes, R. Mch, N. Carr, O. Stava, and G. S. Miller, "Packmerger: A 3d print volume optimizer," *Computer Graphics Forum*, vol. 33, no. 6, pp. 322–332, 2014.

[22] N. Volpato, R. T. Nakashinma, L. C. Galvao, A. O. Barboza, P. F. Benevides, and L. F. Nunes, "Reducing repositioning distances in fused deposition-based processes using optimization algorithms," *High Value Manufacturing*, 2014.

[23] P. K. Wah, K. G. Murty, A. Joneja, and L. C. Chiu, "Tool path optimization in layered manufacturing," *IIE Transactions*, vol. 34, pp. 335–347, 2000.

[24] M. Wojcik, L. Koszalka, I. Pozniak-Koszalka, and A. Kasprzak, "MZZ-GA algorithm for solving path optimization in 3d printing," in *ICONS 2015: The Tenth International Conference on Systems*, 2015.