Dynamic programming approach to adaptive slicing for optimization under a global volumetric error constraint

Andrew Deng, Yasmine Badr, Puneet Gupta,
Electrical and Computer Engineering Department,
University of California, Los Angeles
{puneetg@ucla.edu}

ABSTRACT

In Additive Layer Manufacturing, the layer-based nature of the process results in error in the final object, termed staircase error. This error can be reduced by using a smaller layer thickness, and hence more layers, to print the object. However, in 3D printing with stereolithography, the print time is mostly determined by the number of layers, since the movement of the laser in the X-Y plane is much faster than the movement of the build platform. Therefore, using finer layer thicknesses can significantly increase the print time.

In this work, we propose a novel adaptive slicing algorithm that balances accuracy and print time. The proposed, near-optimal, dynamic programming (DP) based algorithm for adaptive slicing minimizes the number of layers subject to a global volumetric error constraint. Our approach reduces slice count by up to 36% (52%) compared to a state of the art adaptive slicing (uniform slicing) method under the same volumetric error. The results were tested on Formlabsunder the same volumetric error. The results were tested on Formlabs Form1+ SLA-based printers. The print time was improved by up to 32% (53%) for a selection of objects.
Keywords: adaptive slicing, dynamic programming, volumetric error, 3D printing

1. INTRODUCTION

The technology of 3D printing, or layered additive manufacturing, has advanced quickly in the past decade, allowing its use in a wide range of applications ranging from the manufacturing of large objects for use in industrial applications to the construction of small objects for individual use. 3D printers may be used to quickly manufacture an object of arbitrary geometry using a single process, called additive manufacturing (AM). In an additive manufacturing process, objects are built by slowly adding material rather than carving out unnecessary material. Objects are typically built in slices, where material is successively deposited or solidified in layers along the slicing plane. However, due to the slice-by-slice approach to the manufacturing process, error is introduced in each layer because the contour of the object does not always have upright vertical edges . The error in the print from this source is termed staircase error (Figure 2).

The staircase error can be reduced by an increase in the vertical contour resolution, which can be accomplished by reducing the thickness of each slice. Decreasing the minimum thickness means that the 3D printer must print more layers to manufacture the object. The increase in the layer count can lead to a significant increase in the print time. One method of balancing these two factors is adaptive slicing. In most 3D printers, the default configuration is to slice the object uniformly, which means that the object will be sliced at the same vertical precision throughout the object. If high print accuracy is desired, then the entire object must be printed at a small slice thickness. Slicing the object in this way can lead to unnecessary increases in the print time, depending on the object's geometry. For example, if a section of an object has a particularly complex vertical contour but the rest of the object's vertical contour is very flat, then printing at a small slice thickness will improve accuracy in the complex section but will lead to no improvement in accuracy for the rest of the object. Additionally, the simple sections will be printed with thin slices, effectively wasting print time. Since most additive manufacturing technologies allow adjustment in slice thickness during the print, this can be overcome by adaptively changing the slice thickness during the print to match the complexity of the section of the object being printed.

Adaptive slicing has been explored by various authors in the past, and several different methods for obtaining an adaptive slicing scheme have been proposed. This article presents a novel dynamic programming based method for adaptively slicing a 3D object that takes into consideration a global volumetric constraint and returns a near-optimal slicing scheme under that constraint.

2. RELATED WORKS

This section will provide a short summary of work that has already been done on improving print speed for 3D printing, and will also contain a detailed examination of works that focus on quantifying error in 3D printing and works that focus on methods for adaptive slicing.

The print time of a 3D printer is influenced by a variety of different factors. Some work has been done on studying the effect of support structures on print time, and how to modify the construction of support structures to improve print time[1, 2]. Another area of study is the process of orienting the object during the print to improve print time[3]. Additionally, for deposition printers in particular, the improvement of the nozzle path planning process during the deposition of each single layer has been studied[4, 5]. Finally, many different methods for improving print time and print accuracy through adaptive slicing have been proposed. This paper focuses on the latter, which necessitates a discussion of how print accuracy and error are quantified and the details of other adaptive slicing methods.

2.1 Error in 3D printing

There are two main metrics that quantify error in 3D printing. One method is called the cusp height error. This metric quantifies the effect of staircase error on the final printed object by defining the error as the largest distance from the printed object to the vertical contour of the original object for each layer[6]. If the cross section of the missing or additional material volume is depicted as a right triangle, then the cusp height is the length of the altitude drawn from the right-angle vertex to the opposite edge (Figure 1).

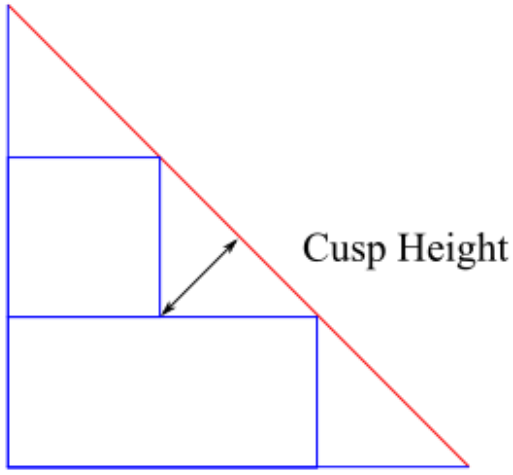

Figure 1: An illustration of cusp height error

While this method of quantifying error estimates the difference between the printed object and the original, the other popular method defines error as simply the volumetric difference between the printed object and the original object. For each layer, the error from the print would be the total volume of material that is either additional or missing in the printed result. The volumetric error metric for optimization in 3D printing has been used in the past in the context of build orientation optimization[7, 8], and it has been noted that using this as the error metric guarantees that the metric will correspond to actual deviation in the printed object while the cusp height error estimate may fail to represent error in some cases[9].

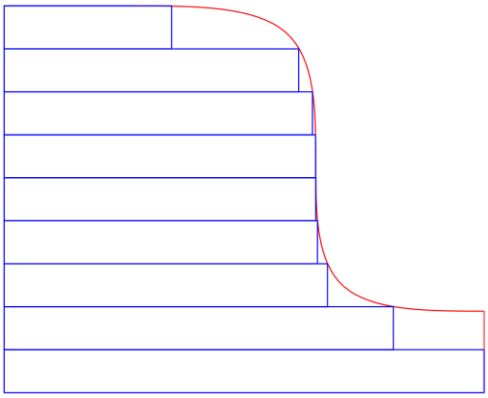

Figure 2: A depiction of staircase error. The original model's vertical profile is in red, while the printed model is in blue. The gaps between the red curve and the blue rectangles represent missing material.

2.2 Adaptive slicing methods

Adaptive slicing for 3D printing has been explored in great detail, with many different proposed methods for determining a slice scheme that results in a fast build time while retaining print accuracy. Dolenc and Mäkelä[6]. give a method for adaptive slicing that analyzes the input CAD model on a local level, and determines whether additional slices are necessary based on geometric heuristics. They use the cusp height error metric to control error on a local level. Suh and Wozny[10] offer a similar solution, with more attention to the vertices of the CAD model. Their method involves sampling points along the cross-section contours and analyzing the local geometry to determine the next slice height. The cusp height error metric is used here as well. Sabourin[11] proposes and implements a solution that utilizes stepwise uniform refinement. In this method, the model is sliced at the maximum thickness to begin with, and each slab is divided further if the error in the slab exceeds a maximum value. Instead of determining the slice thickness across the entire 2D cross section of the object at each height, Tyberg and Bøhn[12] develop a method that slices the model locally in the XY plane. In their method, different areas of the model at each height are sliced differently so that a feature that requires precise slicing does not force a separate feature at the same height from being sliced the same way. Rianmora and Koomsap's[13] adaptive slicing solution directly slices the CAD model using an image-based error metric, instead of the more common cusp-height or volumetric error. Their method involves taking orthographic projections of the original CAD file, finding the edges using image processing techniques, and producing slices by analyzing the contour deviation locally and slicing if the deviation is under a maximum allowable contour deviation. Wang et al.[14] present a method that draws on work done on mesh saliency metrics to provide a method for quantifying how humans perceive objects. Using this metric, they weight the importance of each layer of the model to visual quality (saliency). They then slice the object using the cusp-height error metric, weighted by the computed saliency of each layer. Siraskar et al.[15] use a modified octree boundary data structure (MBODS) to recursively break down an object file into cubes. Cubes were divided based on the slant of the object within each cube as well as the volume of the object contained within each cube, and the slices to be used in the print were determined from the sizes of cubes at each height .

Compared to these works, the approach taken in this paper produces a slicing scheme that considers volumetric error globally rather than locally, and provides a near-optimal slicing scheme that optimizes for build time under a given volumetric error constraint in polynomial time.

3. ADAPTIVE SLICING PROBLEM FORMULATION

The objective of the problem is to produce an adaptive slicing scheme that minimizes build time while staying under the given global volumetric error constraint. Build time can be minimized for two different types of 3D printers, deposition printers and layer solidification printers, in different ways. Deposition printers print an object layer by layer by moving a nozzle that deposits material onto appropriate parts of the object, while layer solidification printing involves solidifying an entire layer of some fluid material into a layer of the desired object. The method discussed in this article applies to layer solidification printers, such as printers that use stereolithography to solidify layers. Since the laser can be made to move extremely quickly during the solidification process, this part of the process takes very

little time, so the time of the print is mostly proportional to the number of slices that need to be printed rather than the overall volume of the object. Therefore, the objective of speeding up printing can be reduced to the objective of minimizing the number of slices for layer solidification printers.

An adaptive slicing scheme is a set of slice thicknesses that sum up to the height of the object to be printed, since the scheme must cover the entire object. The range of possible slice thicknesses present in the scheme must be determined from the 3D printer that this technique is being applied to. With this in mind, the objective of producing an adaptive slicing scheme that minimizes build time while considering a global error constraint becomes a constrained combinatorial optimization, where the goal is to find the combination of slices with the smallest slice count while still meeting the volumetric error requirement.

4. ALGORITHM

A naïve way to solve this optimization problem is to just traverse through all possible slice schemes recursively and find the solution that gives the smallest error. However, since the set of possible slicing schemes is massive, this is not feasible. To solve this in a reasonable time complexity we use dynamic programming. In dynamic programming, the problem is broken down into many subproblems. Each subproblem is solved optimally and its solution reused in construction of the optimal solution of the bigger problem. Since subproblem solutions are memoized, they don't need to be recomputed, making the optimization tractable.

To break down this problem we consider how it would be solved with a simple traversal. Starting with a height H remaining to slice and an error budget of E, choosing a slice of thickness t_i with associated error e_i will reduce the height remaining to $h = H - t_i$ and the error budget to $e = E - e_i$. This would continue until either the error budget is expended (reduced to 0 or a negative value) or the height is reduced to 0. In the case that the error budget is expended, that particular branch cannot meet the error requirement so it is removed. In the case that the height reaches 0, the object has been fully sliced through some sequence of slices thicknesses and an error equal to the sum of all the errors from each slice. Each successful scheme would then have to be compared to find the one with the smallest number of slices.

The performance of this traversal can be improved by defining the subproblems to be different combinations of heights and error budgets, (h, e). Note that if there exist two different sequences of slices that reach this situation there is no reason to pick the sequence with more slices, so only the sequence with the smallest number of slices should be kept as the optimal solution to this subproblem. To extend this to the whole problem, a large 2-dimensional array can be constructed where each cell represents a different subproblem (h, e) and holds information about the number of slices in its optimal solution and the slice thickness used to reach it. One complication with this strategy is that since the error may be any real number, the size of this array would be infinitely large. To get around this, all error values must be rounded to some rounding factor r. This makes the dimensions of the array $\frac{H}{t_{min}} \times \frac{E}{r}$, where t_{min} is the smallest difference between any linear combination of the given slice thicknesses. With the array constructed as such, the problem can be solved with the following recursive algorithm.

Define *solutions*, *errors*
Procedure findOptimalSolution(*height, errorBudget*):
    If *errorBudget* is negative:
        Return bad solution
    If *solutions[height]* exists*:*
        Return *solutions[height*]
    If *height* is 0:
        Return solution with 0 slices and 0 error

    For each possible slice thickness *t* at this height:
        Get error *e* of slice from *errors*
        Record result of: *findOptimalSolution*(*height – t, errorBudget – e)*

    best_solution← result with least slices and most error budget found above with one new slice and the new error

    store *best_solution* in *solutions*

Return *best_solution*

At the beginning of the algorithm, the large array *solutions* is created to store information about solutions to the subproblems as described above. Each entry of the array represents a result of the *findOptimalSolution* procedure and contains information about the number of slices used in the partial solution, the resulting error, and the first slice thickness used to generate the solution. Additionally, the array *errors* is filled with the error information. Next, the recursive procedure *findOptimalSolution* is called on the entire object with the desired error budget.

In the recursive portion of the algorithm, several conditions are first checked for indications that the recursive branch should stop. If the error budget remaining, *errorBudget*, is negative, then this branch cannot produce a slicing scheme that meets error requirements and returns a result with a number of slices greater than the maximum number of slices to indicate that a solution was not achieved. If the array *solutions* already contains data from a recursive instance run with the same parameters, then the stored data is simply returned immediately. If the height parameter given is 0, then this branch of recursion has reached the end of the object and should return a solution with 0 slices and no error. This condition serves as the base case.

If none of the conditions are met, then computation must continue. The algorithm will begin a new branch of recursion for each possible new slice at *height* with the appropriate errors from *errors* and await the solutions. Once all solutions are returned, the result with the smallest number of slices will be taken as the optimal solution at this height and error budget. If multiple solutions have the same number of slices, then the solution with the smallest error cost will be taken. Finally, the algorithm will add one new slice and the appropriate error to the taken solution, and save and return it.

By storing the best intermediate result in the array during execution, all possible configurations of slice thicknesses are being implicitly explored in a much smaller time complexity. In the worst case, the algorithm will compute a result for each entry in the array, so the time complexity is at worst $O\left(\frac{HE}{t_{min}r}\right)$. Note that this neglects the computation of error, which can be performed as a separate step, as explained in the next section. Once the algorithm finishes running, the solution found will have the minimum number of slices possible under the given volumetric error bound. The slicing scheme used to achieve that result can then be recovered by tracing through the array and collecting the slice thicknesses used for each intermediate result.

5. IMPLEMENTATION PROCEDURE

The algorithm was implemented and tested in Python in two separate segments, a volumetric error computation segment and a slice determination segment. This was done so that the error could be precomputed quickly and the algorithm could quickly lookup the volumetric error resulting from placing a slice of a certain thickness at a certain height.

To compute the volumetric error, two different methods were used. The first method was to essentially numerically integrate the error across the plane of each single slice. For each different possible height and slice thickness, a dense grid of points was formed to represent points on the top face of the slicing plane. Then, for each point, a ray was cast outwards to a point outside the object and the number of intersections with the object's geometry was counted to determine whether the point lay inside the object or not. Finally, a ray was cast downwards to the bottom of the slice plane. The length of the ray that lay inside the object and the length that lay outside the ray were calculated, and these lengths were used to determine the contribution to the volumetric error of that section of the slice plane. The contributions of each point in the grid were summed up to obtain the volumetric error of the entire slice. These volumetric error numbers were recorded in an array with dimensions $m \times n$, where m represents the number of heights that it is possible to slice the object at and n represents the number of slice thicknesses.

A second, more approximate, method was also used because the first method was deemed too slow. In this method, the object was broken down into voxels through the use of mesh voxelization software[16, 17]. Then, the error was determined for each slice and height by counting the number of voxels that did not match the voxel value at the top of each slice. These voxels contribute to error because during the 3D printing process, they will be replaced by voxels matching the top voxel. This is illustrated in Figure 3, an image of a cross section of a single voxelized slice. Each square represents a voxel, and the ones in red represent voxels that will contribute to error. The dashed lines surround the area that will actually be printed. The error obtained using this method was compared to the error obtained using the first method and was found to be close. For example, computing the error for a sample object (Figure 4c) using the numerical integration method yielded 19.63 mm3 while the voxelization method yielded 19.91 mm3, a

discrepancy of 1.4%. The discrepancy across a series of sampled objects averaged 1.7%, so the voxelization method was used for volumetric error computation for its speed.

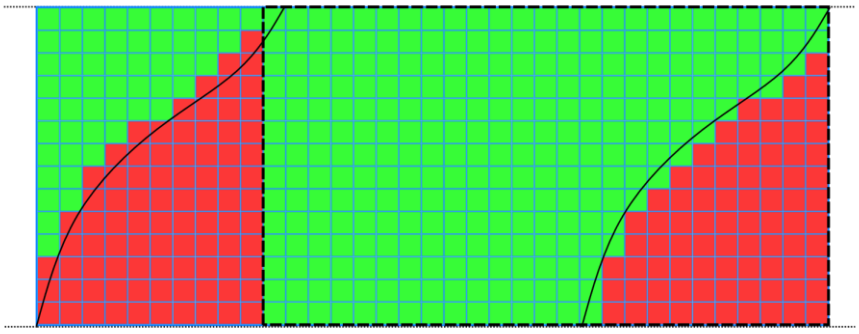

Figure 3: An illustration of the error computation by voxelization.

The slicing algorithm itself was implemented in Python 3 following the algorithm description above. The volumetric error information computed using the methods above was loaded into a global array, and was referenced any time the algorithm needed the error information. After the algorithm was finished running, the slice scheme was recovered by tracing through the memorization array and collecting the slice thicknesses used.

6. EXPERIMENTAL PROCEDURE AND RESULTS

The proposed algorithm was tested against uniform slicing procedures and another state of the art octree-based adaptive slicing method[15] on a Formlabs Form 1+ 3D SLA printer to compare volumetric error and print time. Slicing schemes were gathered from each method for 4 different objects and the total volumetric error for each method was determined using the computed volumetric error tables. For the dynamic programming based method, since the result depended on the volumetric error constraint parameter, multiple constraints were used for each object. The four different benchmark objects are shown in Figure 44. The objects in figure 4(a) and figure 4(b) were obtained from publicly available 3D model repositories[17, 18]. The object shown in figure 4(c) was created for the purposes of this project in the Blender 3D mesh editor, and the object shown in figure 4(d) was obtained from the UCLA School of Dentistry.

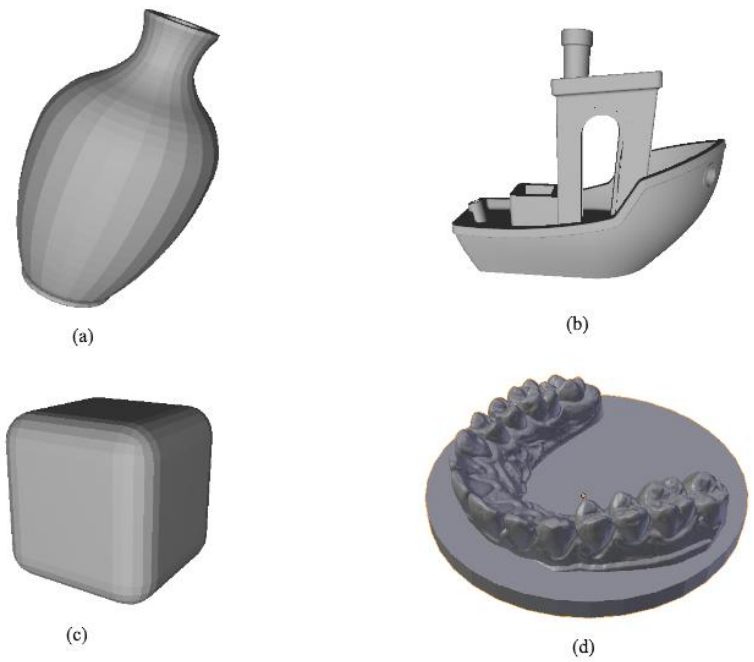

Figure 4 Objects used in experiments. Their volumes are a) 190 mm^3, b) 1940 mm^3, c) 7380 mm^3, and d) 32,660 mm^3.

To determine the print time, the Form 1+ printer was modified to use non-uniform slice schemes through the use of the open source software OpenFL. The different slicing schemes were uploaded to the printer and the print time for one object was recorded by printing it directly. To avoid directly printing each object, the OpenFL software was used to access the print time estimation information. This provided a good estimate of the print time of different slice thicknesses, and this information was used to estimate the print time of the entire object for any given slice scheme. The print time estimated using this method was compared to the print time of the object printed directly and found to agree, so this was used to obtain the estimated print time for each object. Some results are shown in the following table, and plots of volumetric error against print time for each method are shown below as well.

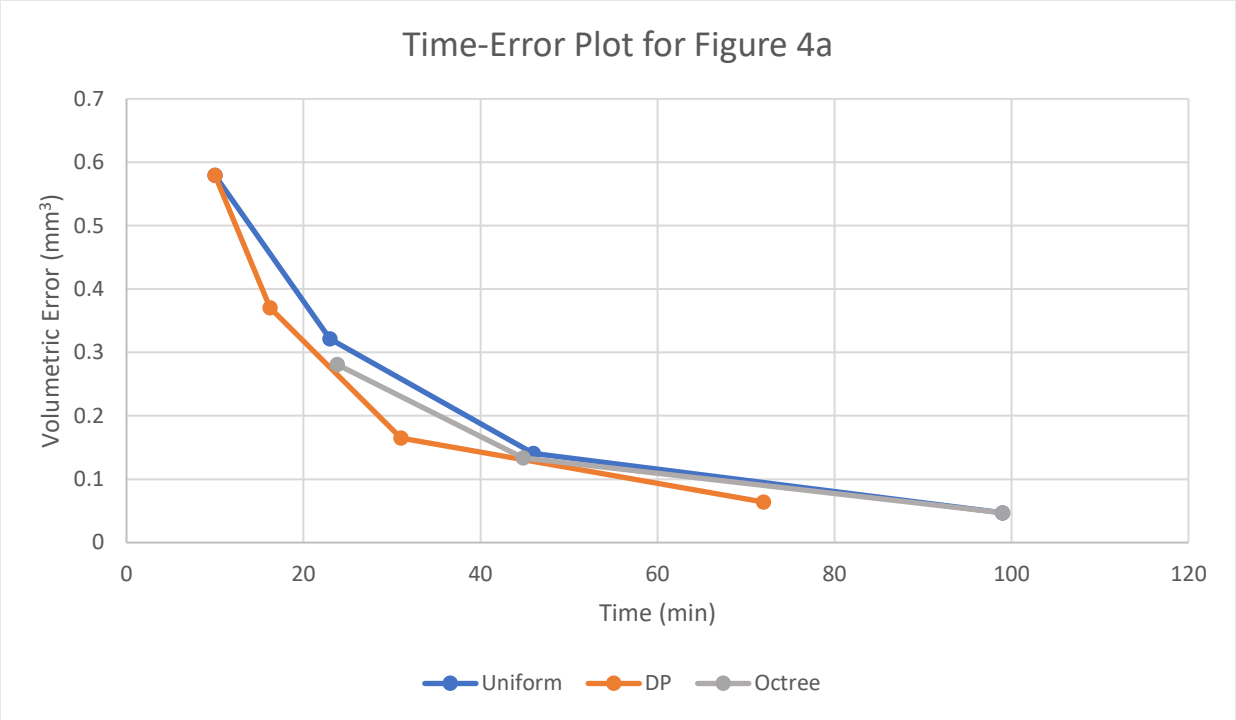

Figure 5 A plot of volumetric error vs. print time for results generated by the various slicing methods used on the object in figure 4a. The object's volume was 190 mm³.

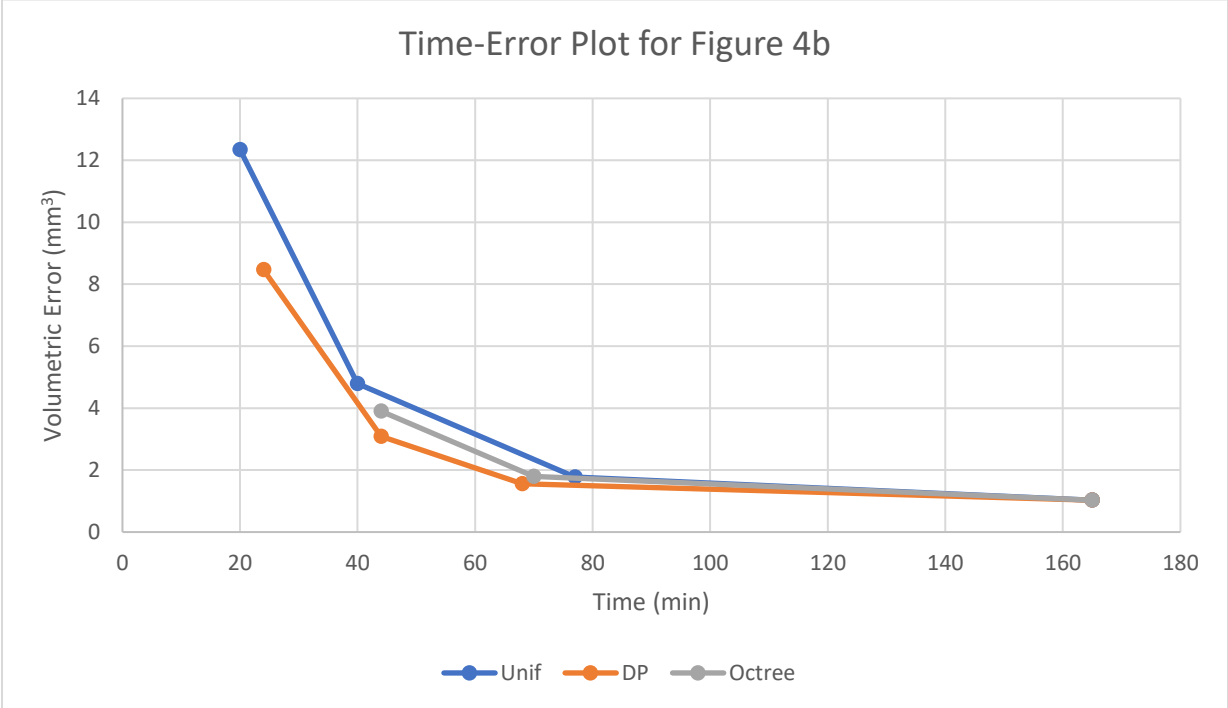

Figure 6 A plot of volumetric error vs. print time for results generated by the various slicing methods used on the object in figure 4b. The object's volume was 1940 mm³.

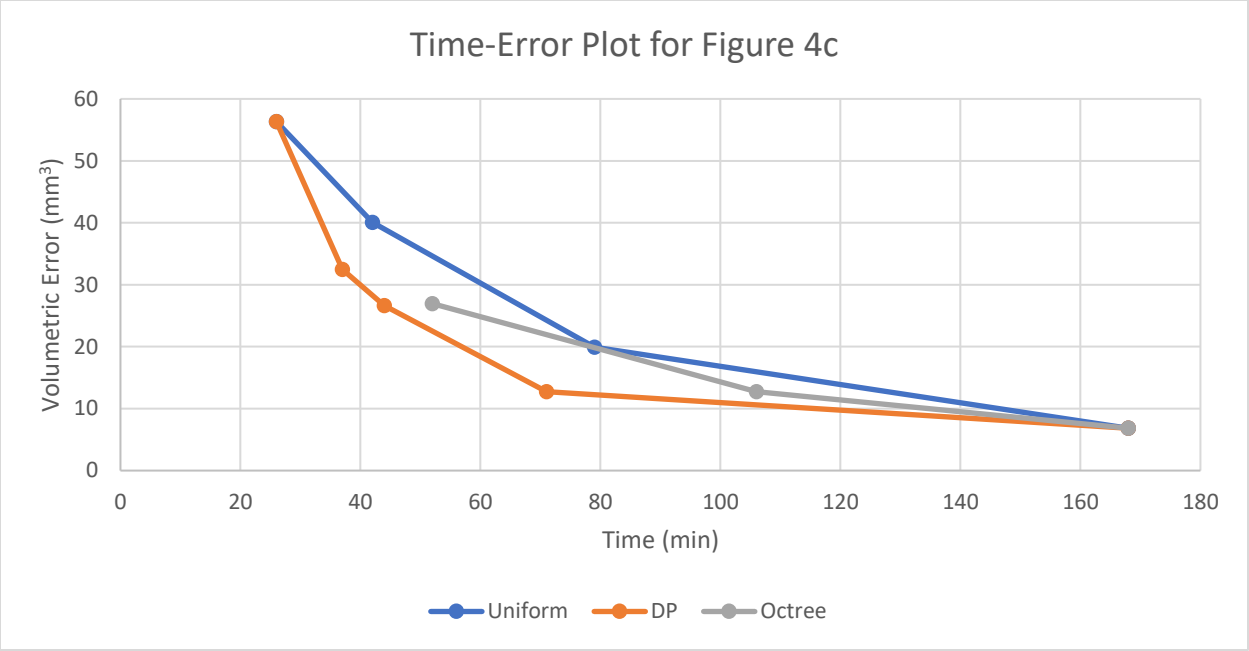

Figure 7 A plot of volumetric error vs. print time for results generated by the various slicing methods used on the object in figure 4c. The object's volume was 7380 mm³.

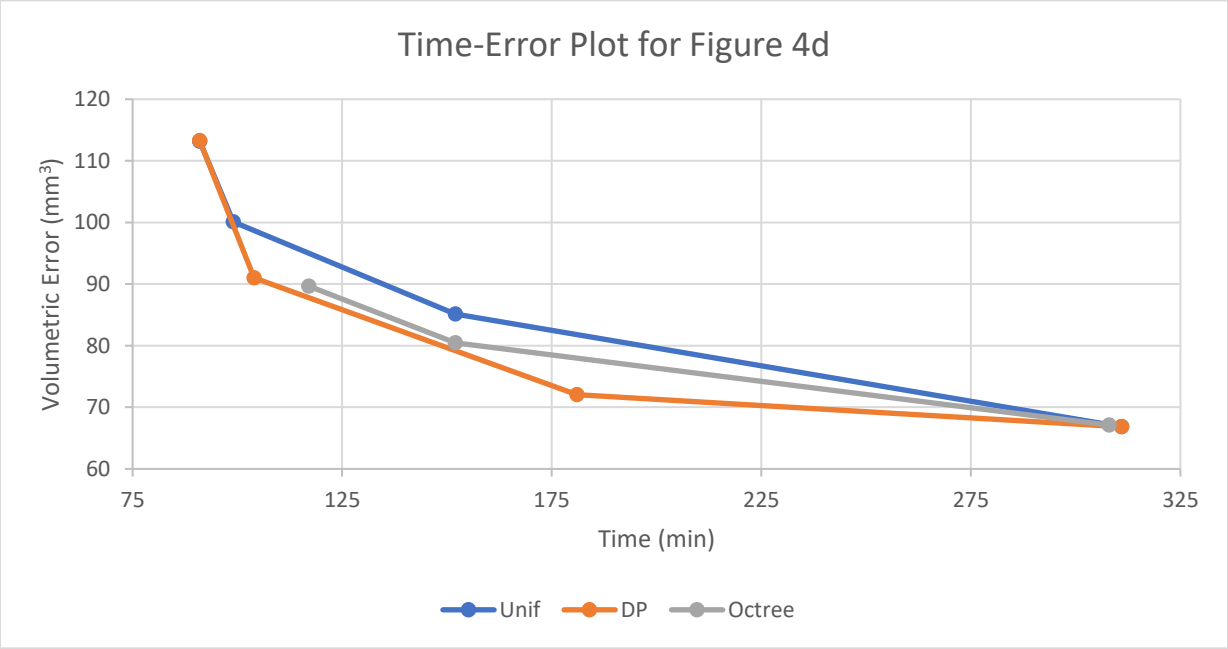

Figure 8 A plot of volumetric error vs. print time for results generated by the various slicing methods used on the object in figure 4d.

Each of the plots above show that the dynamic programming approach achieves either the same or better time and error characteristics when compared to both the uniform slicing and octree adaptive slicing methods. The print-time volumetric error pareto curve for the dynamic programming approach is consistently under the curves that represent the other methods. However, note that at the extremes of each plot, the curves each intersect at the same

points. This occurred because when either algorithm was run with very tight or very loose constraints, they both returned the same scheme as the smallest or largest uniform slice scheme, respectively.

Because the adaptive slicing algorithms usually returned results with both different errors and print times, the individual results were difficult to compare directly. The time-error plots above provide a much better idea of relative performance. Nevertheless, looking in particular at the rounded cube example (Figure 4c) the following comparisons can be made (see table 1). Dynamic Programming approach gives significantly shorter print time with same volumetric error.

Table 1 A comparison of the dynamic programming method and octree method on figure 4c. The volume was 7380 mm^3.

Octree Method			**Dynamic Programming Method**						
Error (mm^3)	**Time (min)**	**Slice Count**	**Error (mm^3)**	**Time (min)**	**Slice Count**	**Error Change**	**Time Change**	**Slice Count Change**	**Model Volume Error Change**
12.73	106	556	12.714	72	356	-0.13%	-32.1%	-35.9%	-0.00023%
26.948	52	248	26.629	44	215	-1.18%	-15.4%	-13.3%	-0.00043%

7. CONCLUSION

This paper introduces a novel method for adaptive slicing that uses dynamic programming to minimize the number of slices needed to print an object when given a volumetric error constraint. The algorithm presented has the advantage of returning the slice scheme with a near-optimal minimum number of slices for the given constraint in polynomial time, due to the implicit exploration of all possible slice combinations with dynamic programming.

The algorithm was tested on a series of objects with a range of volumetric error constraints and compared against a different adaptive slicing algorithm. The results of these two slicing methods were compared to the result of uniformly slicing the objects by each of the different slice thicknesses. For each of the 4 different objects tested, the dynamic programming approach was found to perform significantly better than the octree adaptive slicing algorithm as well as the uniform slicing procedure.

8. ACKNOWLEDGEMENTS

The authors thank Dr. Kumar Shah from the UCLA School of Dentistry for the jaw model that was used for the experimental result.

9. REFERENCES

[1] Han, W., Jafari, M. A., and Wang, C. C., "Support slimming for single material based additive manufacturing," Computer-Aided Design 65, 1-10 (2015).
[2] Reiner, T. and Lefebvre, S., "Interactive modeling of support-free shapes for fabrication", Proc. Eurographics – Short Papers (2016).
[3] Taufik, M. and Jain, P. K., "Role of build orientation in layered manufacturing," International Journal of Manufacturing Technology and Management 27(1-3), 47-73 (2013)
[4] Yang, W., "Optimal path planning in Rapid Prototyping based on genetic algorithm," Proc. Chinese Control and Decision Conference, 5068-5072 (2009)
[5] Fok, K. Y., Ganganath, N., Cheng, C. T., and Tse, C. K., 'A 3D printing path optimizer based on Christofides algorithm", Proc. IEEE-TW, 1-2 (2016)
[6] Dolenc, A. and Mäkelä, I., "Slicing procedures for layered manufacturing techniques," Computer-Aided Design. 26(2), 119-126 (1994).
[7] Masood, S. H., Rattanawong, W., and Iovenitti, P., "Part build orientations based on volumetric Error in Fused Deposition Modelling," Int J Adv Manuf Technol 16(3), 162-168 (2000)
[8] Masood, S. H., Rattanawong, W., and Iovenitti, P., "A generic algorithm for a best part orientation system for complex parts in rapid prototyping," Journal of Materials Processing Technology 139(1), 110-116 (2003)
[9] Taufik, M., Jain, P. K., "Volumetric Error Control in Layered Manufacturing", ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, (2014)
[10] Suh, Y. S. and Wozny, M. J., "Adaptive slicing of solid freeform fabrication processes," Proc. Solid Freeform Fabrication Symposium, 404-411 (1994).
[11] Sabourin, E., Houser, S. A., and Bøhn, J. H., "Adaptive slicing using stepwise uniform refinement," Rapid Prototyping Journal 2(4), 20-26 (1996).
[12] Tyberg, J., and Bøhn, J. H., "Local adaptive slicing," Rapid Prototyping Journal 4(3), 118-127 (1998).
[13] Rianmora, S. and Koomsap, P., "Recommended slicing positions for adaptive direct slicing by image processing technique," Int J Adv Manuf Technol 46, 1021-1033 (2010).
[14] Wang, W., Chao, H., Tong, J., Yang, Z., Tong, X., Li, H., Liu, X. and Liu, L., "Saliency-Preserving Slicing Optimization for Effective 3D Printing", Computer Graphics Forum, 34(6), 148–160 (2015).
[15] Siraskar, N., Ratnadeep P., and Sam A., "Adaptive slicing in additive manufacturing process using a modified boundary octree data structure." Journal of Manufacturing Science and Engineering 137.1 (2015): 011007.
[16] Nooruddin, F. S. and Turk, G., "Simplification and Repair of Polygonal Models Using Volumetric Techniques," IEEE Transactions on Visualization and Computer Graphics 9(2), 191-205 (2003)
[17] Min, P., *binvox*, http://www.patrickmin.com/binvox, 2004-2017
[18] https://grabcad.com/library/vase-57
[19] https://www.thingiverse.com/thing:763622