Slicing Triangle Meshes: An Asymptotically Optimal Algorithm

Rodrigo M. M. H. Gregori*, Neri Volpato†, Rodrigo Minetto* and Murilo V. G. da Silva*
*Departamento Acadêmico de Informática
†Departamento Acadêmico de Mecânica
Federal University of Technology – Paraná (UTFPR)
Curitiba, Brazil
Email: rodrigo.gregori@gmail.com, {nvolpato, rminetto, murilo}@utfpr.edu.br

***Abstract*—Additive Manufacturing, popularly known as "3D printing", is a manufacturing process based on overlapping of flat layers in order to build a physical object. The data for building this object comes from a 3D model, usually represented by a triangle mesh. One of the first stages in this process is to slice the triangle mesh, resulting in 2.5-D contours, representing each one of the layers of the object. There are many strategies for slicing meshes, however most of the current literature is concerned with ad hoc issues such as the quality of the model, specific improvements in the slicing process and memory usage, whereas none of them addresses the problem from an algorithmic complexity perspective. While current algorithms for slicing run on $\mathcal{O}(n^2 + k^2)$ or $\mathcal{O}(n^2 + n \log n\overline{k})$ for n triangles and k planes, the algorithm proposed in this paper runs on $\mathcal{O}(n\overline{k})$, where $\overline{k}$ is the average number of slices cutting each triangle, what is asymptotically the best that can be achieved under certain common assumptions.**

***Keywords*-process planning, triangle meshes; slicing, interval tree, stabbing problem**

I. Introduction

Triangular meshes are widely used to represent geometric models in many applications, such as Geological Modeling, Computer Aided Design (CAD), Computer Graphics, and Addictive Manufacturing.

Additive Manufacturing (AM) is a technique consisting of building an object by laying down successive flat layers of material. This technique gained a great deal of popularity recently and it is usually known as 3D printing. The data for building this object come from a three-dimensional geometric model, obtained via a CAD system or from computerized tomography and magnetic resonance.

Before the physical prototyping process, the geometric model undergoes a *Process Planning*, a procedure that vary according to the complexity of the model or the prototyper technology. Those stages include: correctly positioning the object in space, in order to achieve better quality or faster building; slicing the geometric model (the problem that we address in the paper), calculating supports for building the models among other engineering issues.

The model is usually tessellated into a triangle mesh, and the *de facto* industry format for storing it is the STL (from StereoLithography) model. This format is very popular due to its simplicity [1], however, since a mesh vertex can be shared by many triangles this might lead to redundancy.

The mesh must be sliced in order to obtain the contour information, and each slice is "printed" by the prototyper and the overlapping of all slices composes the final object. In order to improve the quality of the printed model and spend less time in the process, modern slicing procedures make use slices of variable thickness, this approach is also known as adaptive slicing. An extensive review on adaptive slicing can be found in [2].

The slicing problem has been addressed in the rapid prototyping research community, but most of the current literature is concerned with ad hoc (but still relevant) engineering issues such as the quality of the model, specific improvements in the slicing process and memory usage, whereas none of them addresses the problem from an algorithmic time complexity point of view, even though the slicing process can consume up to 60% of the entire process planning time [3]. We propose an algorithm for the adaptive slicing problem which performs optimally, unless the number of slices k is $o(\log n)$, for n triangles, which is only the case for uninteresting cases of small models made up of only few layers.

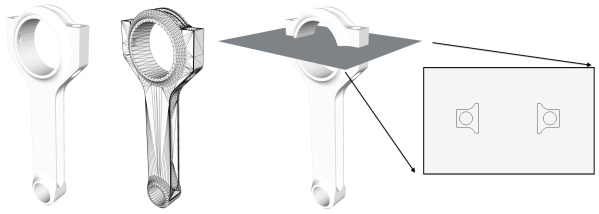

Figure 1. Selected steps in the Additive Manufacturing Process Planning: From left to right, a CAD model, a tessellated (STL format) model, slicing the mesh and contour generation for reconstructing a layer of the manufactured object.

II. Triangle Mesh Slicing

We give in this section a brief overview of the strategies for the problem of triangle mesh slicing. Interestingly, even the trivial solution, i.e., looping through all the triangles in the mesh against all cutting planes and checking whether

there is a plane/triangle intersection, appeared in the literature [4]. There has been also some variations on this naive approach via parallelism [3] and optimization of memory usage [5], [6].

A more sophisticated approach appears in [7], where the algorithm takes a "snapshot" for each z-coordinate of a given cutting plane and check for edges (instead of triangles) to be "sliced". In a different approach, in [8] the strategy is to group together triangles according to their minimum and maximum z coordinates so that for a given plane, a search for intersecting triangles can be faster. The approach in [9] only deals with slices of fixed width, which is an algorithmically simpler problem than the one we are dealing in this paper (i.e., the "adaptive" version of the problem). The idea in their work is to loop through every triangle, and given its min/max z coordinate, simply directly calculate every intersecting plane achieving optimality for this simpler version of the problem.

From a broader perspective, we can approach the problem that we are dealing in two different manners:

(1) For every cutting plane, make a query in the set of all triangles and retrieve intersecting triangles. This is the approach used in all previous works[1] [3]–[8]. In this scenario, for each query, an optimal algorithm ideally would retrieve only the triangles sliced by the current plane. None of the mentioned algorithms achieves such optimality.

(2) For every triangle, query the set of all planes and retrieve intersecting planes. Our algorithm follows this second approach making use of an interval tree data structure [10] for storing line segments that represent triangles. The algorithm performs an in-order traversal in this tree[2], taking advantage of a property of the triangle mesh (the object is closed 2-manifold) to progressively retrieve only planes that intersect the current triangle (i.e., it is optimal).

After all the intersections are obtained, the next step is the contour assembly, i.e., given a plane, the algorithm should gather all segments from the intersection of this plane with the triangles in the mesh and assemble the polygon that make up the contour of the corresponding slice. A "head-to-tail" sorting is commonly used [4], [11] for this purpose. We use a hash table in order to solve the problem in linear time.

A. Trivial slicing

The simplest algorithm to tackle the stated problem, outlined in Figure 2, is to test every mesh triangle against all the cutting planes. In order to recover the contour of the slices, the triangle–plane intersection segments of every slice stored in line 6 of the algorithm can be sorted from head-to-tail (line 7) in $\mathcal{O}((k\overline{n}\log\overline{n}))$ time, where k is the number of slices and $\overline{n}$ be the average number of triangles per slice. For n triangles, the total time complexity of this approach is $\mathcal{O}(k(n+\overline{n}\log\overline{n}))$.

```
1: TRIVIAL-SLICING (T[1...n], P[1...k])
2:   S[1...k] ← ∅;                          ▷ Shape slices.
3:   for p ← 1 to k do
4:     for t ← 1 to n do
5:       if (T[t] ∩ P[p]) ≠ ∅ then
6:         S[p] ← S[p] ∪ INTERSECTION (P[p], T[t]);
7:   BUILD-SLICES (S[1...k]);
```

Figure 2. The trivial slicing algorithm

B. Sweep Plane Slicing

The sweep plane slicing strategy, proposed by McMains and Séquin in [7], consists of a "virtual" plane sweeping through the triangle mesh in a bottom-up fashion. A *status data structure* keeps a circular linked list with pointers to the edges that are being intercepted by the sweeping plane at a given moment. Each time the plane reaches a vertex or a z-coordinate of a slice, an event is triggered. The event is either a snapshot of the edges being cut at that moment or the rearrangement of the status data structure, in the case of the sweep plane reaching a vertex.

The idea is not complex, but the steps in the vertex processing are quite involved. Whenever a vertex is reached, it is necessary to figure out which edges are intercepted by the sweeping plane after the vertex has been overtaken. This procedure has a worst case complexity of $\mathcal{O}(n^2)$ for complex models, but in average the complexity is $\mathcal{O}(n\log n)$.

The algorithm is implemented using a new data structure for storing topological data. The triangle boundaries are represented by a structure called *edge uses*. Building up this structure is achieved in $\mathcal{O}(n\log n)$ complexity via hashing.

C. Triangle Grouping

A different approach, aiming to minimize failed triangle queries, group together triangles at similar z-coordinates [8]. The slicing engine is based on a "facet processor", responsible for grouping triangles according to their minimum and maximum z coordinates (z_{min} and z_{max}).

In this algorithm, the triangles are sorted by z_{min} coordinates in $\mathcal{O}(n\log n)$ time and then grouped so that triangles with the same z_{min} value are clustered together. Each group can be divided into sub-groups, organized by z_{max} values. Triangles sharing z_{max} values are stored in the same subgroup. The implementation details are not shared, but each group could be implemented with a binary search tree so that this construction can be done in $\mathcal{O}(n\log n)$.

The algorithm processes the triangles using a "key characteristic identifier" in order to calculate slice thickness of the

[1]Except in [9], however, as mentioned before, their work does not deal with adaptive slicing.

[2]Alternatively we could also sort the triangles in lexicographical order in an array and then probe it linearly.

Table I
IN THIS TABLE n AND k ARE THE NUMBER OF TRIANGLES AND THE NUMBER PLANES RESPECTIVELY, $\overline{n}$ IS THE AVERAGE NUMBER OF TRIANGLES INTERSECTING ONE PLANE AND $\overline{k}$ ARE THE AVERAGE NUMBER OF PLANES INTERSECTING ONE TRIANGLE.

Algorithm	Memory Construction	Worst case slicing	Contour assembly	Worst case total (assuming $k = \Omega(\log n)$)
Trivial	$\mathcal{O}(n)$	$\mathcal{O}(nk)$	$\mathcal{O}(\overline{n} \log \overline{n})$	$\mathcal{O}(nk + \overline{n} \log \overline{n}k)$
Triangle Grouping	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2 + n \log n\overline{k})$	$\mathcal{O}(\overline{n})$	$\mathcal{O}(n^2 + n \log n\overline{k})$
Sweep Plane	$\mathcal{O}(n \log n)$	$\mathcal{O}(k^2 + n^2)$	$\mathcal{O}(\overline{n})$	$\mathcal{O}(n^2 + k^2)$
Incremental Slicing (our method)	$\mathcal{O}(n \log n)$	$\mathcal{O}(n\overline{k})$	$\mathcal{O}(\overline{n})$	$O(n\overline{k})$

```
1:  INCREMENTAL-SLICING (T[1...n], P[1...k])
2:    S[1...k] ← ∅;                    ▷ Shape slices.
3:    next ← 1;
4:    for t ← 1 to n do
5:      p ← next;
6:      flag ← true;
7:      while (P[p] ∩ T[t]) ≠ ∅ do
8:        S[p] ← S[p] ∪ INTERSECTION (P[p], T[t]);
9:        if (P[p] ∩ T[t + 1]) ≠ ∅ and flag then
10:         next ← p;
11:         flag ← false;
12:       p ← p + 1;
13:     if flag then
14:       next ← p;
15:   BUILD-SLICES (S[1...k]);
```

Figure 3. Incremental slicing algorithm

slices, a problem that we are not dealing, since we assume that the z-coordinate of each slice is already calculated.

The slices can be obtained the same way as in the trivial algorithm with time complexity of $O(kn \log n)$. The whole adaptive slicing process is at best $\mathcal{O}(n^2)$, even though, later, the authors take advantage of their data structure and the contour assembly can be done in approximately linear time.

D. Algorithm Complexity Summary

Table I summarizes the algorithm complexity of the reviewed algorithms and our method (described in next section) under the reasonable assumption of $k = \Omega(\log n)$.

III. INCREMENTAL SLICING: AN ASYMPTOTICALLY OPTIMAL ALGORITHM

The idea behind our algorithm is to represent each triangle t_i in the mesh by an interval $l_i = [z^i_{min}, z^i_{max}]$, where z^i_{min} and z^i_{max} are the lowest and highest z-coordinates of t_i in the mesh. So the set of n triangles in the mesh can be seen as a set of n intervals (See Figure 4).

We store the intervals in an interval tree [10] and the set of k planes from bottom to up in an array P of z-coordinates of these planes. We note that one way of checking for plane-triangle intersection is to model the problem as a *stabbing problem* [12], [13], i.e., given a value z, and a set $S = \{l_1, ..., l_n\}$ of intervals, retrieve each interval in S containing z. Using an interval tree, this query can be done in $\mathcal{O}(\log n + \overline{n})$, where $\overline{n}$ is the number of intervals containing z[3]. In this setting, if one aims for optimality, we note that a query returning $\overline{n}$ triangle can at best be done is $\mathcal{O}(\overline{n})$. This bound is only achieved under certain assumptions that do not hold in our problem in [14].

However, we achieve optimality turning the problem around, more precisely, querying for planes intersecting a given interval (i.e., a triangle). Looking the problem from this perspective, we achieve optimality if for every interval l_i, we manage to retrieve all planes intersecting l_i in time $\mathcal{O}(\overline{k_i})$, where $\overline{k_i}$ it the number of planes intersecting l_i, i.e., for n intervals, optimality means $\mathcal{O}(n\overline{k})$, where $\overline{k}$ is the average number of planes intersecting each triangle. We achieve this in the following way: We traverse the interval tree in-order (this can be seen as probing the intervals lexicographically by z^i_{min} and z^i_{max}) and make sure that every time that we reach an interval l_i in the traversal, we already have an index to the first plane in the array P intersecting l_i. So we only need to search P linearly from this index checking for intersection and retrieving the desired $\overline{k_i}$ planes. A crucial point is that while retrieving the $\overline{k_i}$ planes, we look ahead for the next interval l_{i+i} and check which of these $\overline{k}$ planes is the first to intersect l_{i+1}. We note that for a general set of intervals there is no guarantee that l_{i+1} intersects one of the $\overline{k_i}$ planes, but since these intervals come from triangles in a 2-manifold mesh (otherwise "printing" the mesh is not well defined), we have this guarantee.

We describe this algorithm in Figure 3 assuming for clarity that the input is a sorted set T of n triangles (instead of explicitly traversing the tree) and an array P of k planes. Also for clarity, instead of performing a linear time preprocessing step, we assume that that the first triangle and the first plane (i.e., $T[1]$ and $P[1]$) intersects and we also do not check the array bound violation in step 9 when $t = n$. The main idea behind the algorithm is that for each triangle $T[t]$, in step 5 variable p holds the index for the first plane intersecting $T[t]$. The loop at step 7 probes P retrieving every plane intersecting $T[t]$ and at the same time a look ahead in step 10 saves the first plane intersecting $T[t + 1]$.

Finally, for each plane we build the polygon (or polygons)

[3]Note that the time complexity depends also on the size of the output. This model, known as output sensitive complexity model, is commonly used in such problems [14].

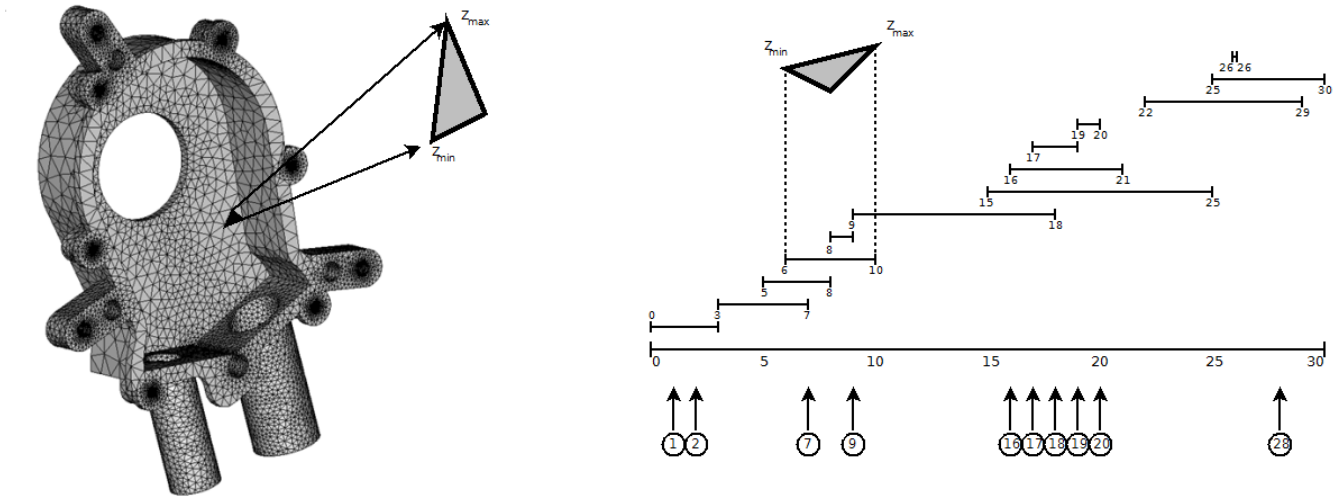

Figure 4. A triangle mesh and highlighted triangle which is represented by an interval. In the right we give an example of 13 intervals sorted in lexicographic order and highlight the interval $[6, 10]$. This interval is cut by planes at z-coordinates 7 and 9, indicated in the bottom of the figure.

that make the corresponding slice. Instead of sorting the line segments from head to tail we can make use of hashing to assemble the polygon in linear time.

IV. Conclusion

We have proposed an optimal algorithm for the adaptive slicing problem modelling the slicing problem as a sequence of stabbing queries on intervals in the particular case where there is always intersection between two consecutive intervals (a property of 2-manifold triangle meshes). We compared asymptotically our algorithm against three other methods described in the literature. As future work, we plan to compare the execution time of these algorithms in a real additive manufacturing process.

References

[1] J. Hiller and H. Lipson, "STL 2.0: A Proposal for a Universal Multi-Material Additive Manufacturing File Format," in *Solid Freeform Fabrication Symposium*, no. 1, 2009, pp. 266–278.

[2] 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.

[3] C. Kirschman and C. Jara-Almonte, "A parallel slicing algorithm for solid freeform fabrication processes," in *Solid Freeform Fabrication Symposium*, 1992, pp. 26–33.

[4] K. Chalasani and B. Grogan, "An algorithm to slice 3D shapes for reconstruction in prototyping systems," in *ASME Computers in Engineering Conference*, 1991, pp. 209–216.

[5] S. Choi and K. Kwok, "A tolerant slicing algorithm for layered manufacturing," *Rapid Prototyping Journal*, vol. 8, no. 3, pp. 161–179, 2002.

[6] M. Vatani, A. Rahimi, and F. Brazandeh, "An enhanced slicing algorithm using nearest distance analysis for layer manufacturing," *Proceeding of World Academy of Science, Engineering and Technology*, no. 25, pp. 721–726, 2009.

[7] S. McMains and C. Séquin, "A coherent sweep plane slicer for layered manufacturing," in *Proceedings of the fifth ACM symposium on Solid modeling and applications - SMA '99*. New York, New York, USA: ACM Press, 1999, pp. 285–295.

[8] K. Tata, G. Fadel, A. Bagchi, and N. Aziz, "Efficient slicing for layered manufacturing," *Rapid Prototyping Journal*, vol. 4, no. 4, pp. 151–167, 1998.

[9] X. Huang, Y. Yao, and Q. Hu, "Research on the Rapid Slicing Algorithm for NC Milling Based on STL Model," in *Asia Simulation Conference (AsiaSim)*, 2012, pp. 263–271.

[10] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Santa Clara, CA, USA: Springer-Verlag TELOS, 2008.

[11] S. Choi and F. Kwok, "A Memory Efficient Slicing Algorithm for Large STL Files," in *Proceedings of Solid Freeform Fabrication Symposium*, 1999, pp. 155–162.

[12] H. Edelsbrunner, *Dynamic Data Structures for orthogonal intersection queries. [Mit Fig.]*, ser. Forschungsberichte: Institut für Informationsverarbeitung. Inst. f. Informationsverarbeitung, TU Graz, 1980.

[13] X. C. P. A. R. Center and E. McCreight, *Efficient algorithms for enumerating intersecting intervals and rectangles*. Xerox, Palo Alto Research Center, 1980.

[14] J. M. Schmidt, "Interval stabbing problems in small integer ranges," *Algorithms and Computation*, no. Grk 1408, pp. 1–10, 2009.