An Efficient Triangle Mesh Slicing Algorithm for All Topologies in Additive Manufacturing

Bethany King (1), Allan Rennie (1) , Graham Bennett (2)

(1)Lancaster University, Lancaster, UK

(2)Euriscus Ltd, Chesham, UK

Corresponding author email address: b.a.king@lancaster.ac.uk

Title: An Efficient Triangle Mesh Slicing Algorithm for All Topologies in Additive Manufacturing

Abstract

To date, slicing algorithms for additive manufacturing are most effective for favourable triangular mesh topologies; worst case models, where a large percentage of triangles intersect each slice-plane, take significantly longer to slice than a like-for-like file. In larger files, this results in a significant slicing duration, when models are both worst case and contain more than 100,000 triangles. The research presented here introduces a slicing algorithm which can slice worst case large models effectively. A new algorithm is implemented utilising an efficient contour construction method, with further adaptations, which make the algorithm suitable for all model topologies. Edge matching, which is an advanced sorting method, decreases the number of sorts per edge from *n* total number of intersections to two, alongside additional micro-optimisations that deliver the enhanced efficient contour construction algorithm. The algorithm was able to slice a worst-case model of 2.5 million triangles in 1025s. Maximum improvement was measured as 9,400% over the standard efficient contour construction method. Improvements were also observed in all parts in excess of 1000 triangles. The slicing algorithm presented offers novel methods that address the failings of other algorithms described in literature to slice worst case models effectively.

Key words: Additive Manufacturing; Slicing Algorithm; Efficiency; Computational Geometry; Rapid Prototyping

Paper Type: Research paper

1 Introduction

Additive Manufacturing (AM) can be defined as a technology where a Three-Dimensional (3D) object is constructed by the sequential creation of Two-Dimensional (2D) layers [1]. The creation of components can be performed using a range of methods and materials, however all AM processes consist of three distinct stages: (i) Construction of a digital model; (ii) Application of pre-processing algorithms, converting the model into 2D layers then generating the machine toolpath [2] and (iii) Creation of the part by either depositing or fusing material to the preceding layer. The benefits of AM include increased design possibilities over subtractive manufacturing and increase in efficiency and cost in small volumes [3].

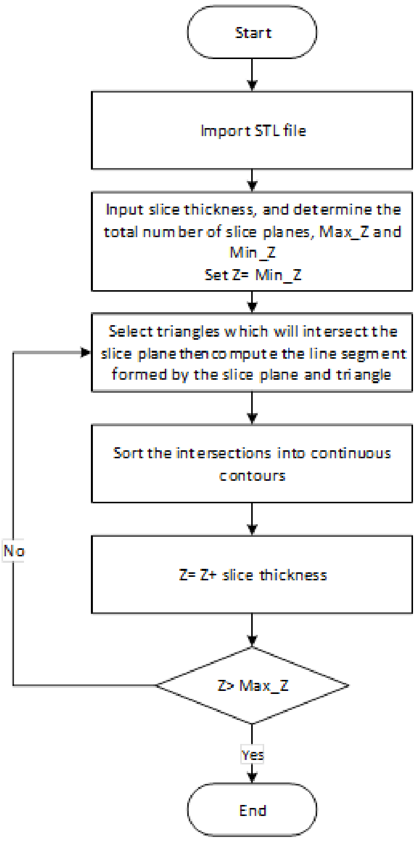
Of the three primary file formats for AM (*.STL, *.AMF, *.3MF) [4-5] all construct geometry using triangular meshes. Meshes in AM always consist of tessellated triangles which connect at the vertices, each vertex defined as a 3D floating point coordinate and are ordered counter-clockwise when observing the part from the outside [6], an associated outward facing normal is attributed to each triangle, which can be utilised during slicing or when graphically rendering the part [7]. As technology has advanced and the resolution and accuracy of machines has improved [8] the meshes in AM files required to capture the more detailed geometric features have increased in complexity and become finer [9]. The slicing algorithm required to convert modern AM models into 2D contours must continue to improve, to slice what was once considered exceptionally large files efficiently.

Part models that are worst case from a slicing algorithmic perspective are those containing a large percentage of triangles intersecting on any given layer. The slicing process consists of two operations calculating the intersections between the triangles and the slice plane and then sorting the intersection into contiguous contours. Worst case parts are particularly difficult to slice due to the sorting process, increasing in duration exponentially by each additional triangle in the layer. The research presented here builds on the Efficient Contour Construction (ECC) method [10] that exploits the triangular mesh format adding features that address the inability to handle worst case parts effectively.

2 Review of related works

Slicing is the process of converting the 3D model into a series of layers containing the 2D perimeter boundaries characterised by a closed loop of connected points [11]. Xu *et al.* [12] offer a basic description of the stages involved in the slicing process (Figure 1), consisting of calculating all the intersections for one slice plane then sorting them into a continuous contours method that works very well for simple geometries but becomes highly inefficient as complexity increases, due to the consideration of triangles that don't intersect with the slice plane. Tian *et al* [13] describe a method where pre-grouping triangles according to whether they fall into a collection of slice planes using a binary search to reduce such considerations. Whilst a significant increase in efficiency can be observed, it would be better if consideration of redundant triangles could be eliminated entirely.

In the standard slicing model (Figure 1), sorting of the generated intersections relies on comparison of end points of the generated line segment when intersecting the triangle with the slice plane, as described by Steuben *et al.* [14] taking the form of a connected graph search [15] causing false matches due to models where more than two triangles converge on one point (Figure 2). This causes the algorithm to either fall into a continuous loop or produce a failed output, for which a better method of sorting is required.



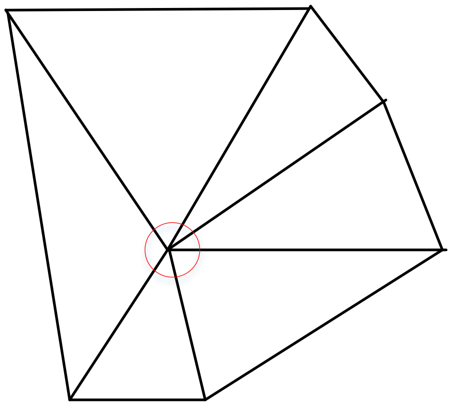


Figure 1: Basic slicing algorithm flowchart

Figure 2: Mesh with shared point

Typically, layer thickness is constant during the slicing process however, there are a number of examples for adaptive slicing [16-18] where the layer height is decreased when slicing regions of high detail or increased when there are less geometric features to be captured. These methods can increase the efficiency of the slicing process; however, use is only appropriate where one model is produced per build cycle. In larger AM machines, such as Selective Laser Sintering (SLS), Stereolithography (SL) or Selective Laser Melting (SLM), where conventionally, multiple models are tessellated into the build area. Increasing or decreasing the slice depth for one model will likely be to the detriment of other models on the layer. Li & Xu [19] acknowledge that adaptive slicing is primarily useful for Fused Deposition Modelling (FDM) and can therefore not be considered suitable for a universally efficient slice engine.

Several slicing algorithms produce an optimal output for specific methods of AM. Ding *et al.* [20] suggests slicing in multiple orientations for wire-feed based AM, primarily with the goal of increasing part integrity and minimising support structures. There is similar research attempting to optimise the slicing process for powder bed fusion technology [21] however similar, more significant improvements in this regard can be seen in the optimising model orientation and arrangement [22-24]and should therefore not be the responsibility of the slicing algorithm.

Combining the slicing algorithm with tool path generation [25] can improve the efficiency of the overall process by removing the need to write to an intermediary slice file, but limits the possibilities of the output of the algorithm to the specific application, due to the varying nature of the toolpath input format. There have been a number of efforts to compensate for low quality models, containing errors or incorrect geometric features using the slicing process; Zhao *et al.* [26] aimed to reduce the error caused by discretising the CAD model into the triangular mesh file using contour approximation and Luo & Wang [27] similarly aimed to minimise the impact of defects such as cracks and overlapping edges in the model during the slicing process.

Zhang's [10] ECC algorithm presents a comprehensive universal slicing algorithm that is both time and memory efficient; their method which exploits the clockwise nature of a triangular mesh, allowing for only one intersection per slice plane per triangle to be computed, reducing the memory requirement of the slicing algorithm by half. Additionally, the dynamic sorting method is utilised where intersections are inserted directly into the contour as they are calculated rather than using an intermediary data structure to hold the unsorted line segments, again reducing the memory requirement of the algorithm. The sorting method enables lines to be connected so that only the start and end of the connected line segments need to be checked, drastically reducing the number of sorts and therefore the amount of time taken for sorting than the end-to-end line segment sort detailed in Figure 1.

3 Methods

3.1 ECC algorithm implementation

Zhang *et al.* [10] offers a robust ECC algorithm, which is efficient for some part geometries. The algorithm relies on calculating the intersections for each triangle from top to bottom along either the longest edge, or the two shorter edges (Figure 3). The decision is reliant on the order the vertices of the longest edge appear in each triangle. On calculation of the intersection, it is stored in an Intersection Node (IN), which contains the 2D coordinates of the intersection, an Edge Pointer (EP) containing the memory address of the vertices of the two edges of the triangle intersecting the slice plane, and next and previous pointer, which locates a following or preceding IN in the list respectively (Figure 4). A series of linked INs are held in an Intersection Linked List (ILL) data structure, which contains a pointer to the subsequent ILL and a pointer to the first and last element in the list.

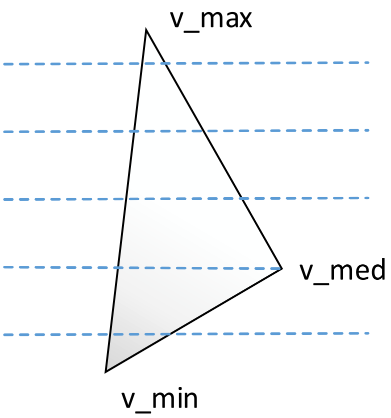


Figure 3: Mesh triangle with slice planes

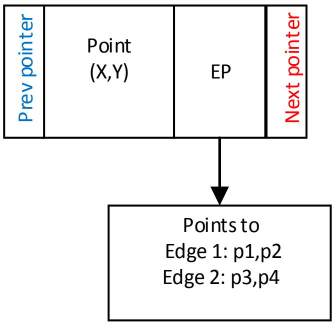


Figure 4: Graphical representation of IN data structure

Following the creation of the IN, it must be inserted into an ILL. All the existing ILLs first and last element's edge pointers are compared with the edge pointer of the IN for insertion. The IN is then inserted according to the following senarios:

1. There are no existing ILLs, the IN is the first calculated intersection on that layer, the IN is inserted in a new ILL;
2. One match is found with the first IN in an ILL, the IN is inserted at the front of the list;
3. One match is found with the last IN in an ILL, the IN is inserted at the back of the list;
4. Two matches are found, in separate lists, at the first element in one ILL and the last element in a second ILL, the IN connects the two lists, and the second list is deleted;
5. Two matches are found in the same list, the IN is inserted at the back of the matched list, this indicates the matched list has been completed;
6. No match is found in any of the existing ILLS, the IN is inserted in a new ILL.

Once the IN has been inserted into an ILL, the following intersection on the edge is calculated and sorted. Once all the intersections on the triangle have been computed and inserted into ILLs, the subsequent triangle is considered until all triangles in the mesh have been considered and slicing reaches completion. The data in the ILLs can then be written into a slice file format, or the ILL list format can be used directly for generation of the toolpath.

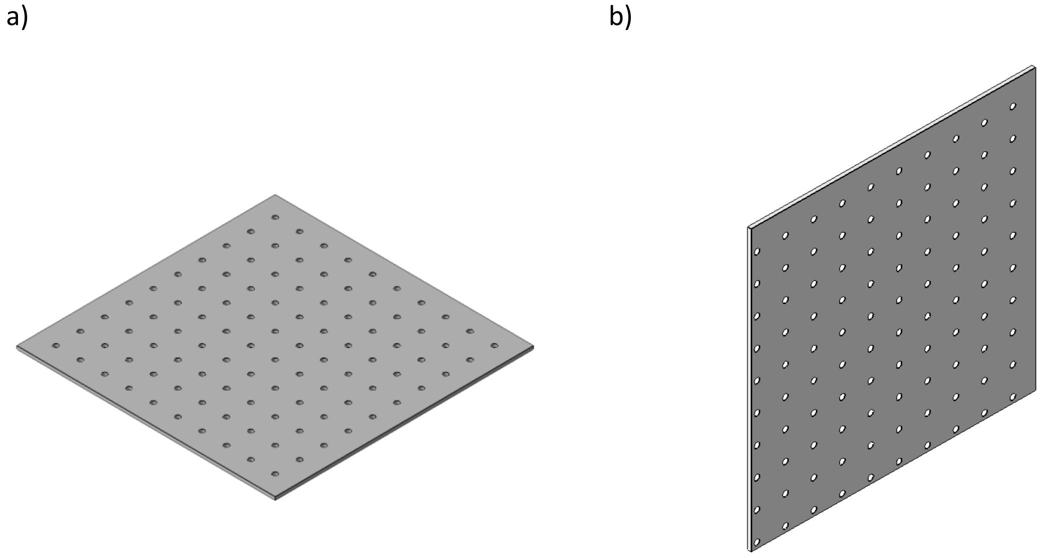


Figure 5: a) Test sheet with 100 holes with 102812 triangles and dimensions 265x265x3mm; b) same test sheet rotated 90^{o}

Upon implementation of the ECC algorithm and a traditional strategy based on the flowchart detailed in Figure 1 [12], in the C++ language comprehensive testing was performed on the *.STL files shown in Figures 5 and 6, the results are given in Table 1. The ECC algorithm shows improvements over the conventional slicing method between 9000-1150% for all three parts. The result was especially impressive for Figure 6, taking under 575ms to slice the part. However, in both the conventional and the ECC method there is a disparity between the slice duration for Figure 5a and Figure 6 despite consisting of a similar number of triangles, took in excess of 237,000ms and 2,752,181ms for the ECC and conventional algorithms respectively – 413 times longer to slice for the ECC method and 494 times longer for the conventional algorithm.

Figure 5b slice time is 20.9 and 19.2 times less than that of Figure 5a for the ECC and conventional algorithm respectively, indicating the direction of slicing interacting with the topology of the part has a significant impact on the effectiveness of the both algorithms. A possible solution is to analyse parts and orient them in a way that is optimal for the slicing algorithm, however this does not present a good result as the optimal orientation for slicing is unlikely to be the optimal orientation for building the part [28] An algorithm capable of slicing the parts efficiently, regardless of the orientation, is essential.

Analysis of the topology of Figure 5a indicated the average percentage of triangles intersecting on each layer is 33% equating to 34,000 triangles, meaning the part can be identified as worst case, whereas for Figure 5b an average of less than 2% or 2056 triangles, intersect on each layer. The total number of intersections of the entire part remains the same for both cases. It can be derived that the sorting procedure when a large number of triangles are present per layer is the cause of inefficiency.

Figure 6: Chess Rook containing 93,930 triangles, dimensions: 31.75 x 31.75 x 53.15mm

Table 1: Performance of ECC and traditional slicing algorithm, all parts were sliced at 0.1mm slice thickness on a 64 bit system

Model	**Size (L, W, H mm)**	**Triangles**	**Conventional Slicing algorithm (ms)**	**ECC Time (ms)**
Test sheet with 100 holes (Figure 5a)	256 x 256 x 3	102,812	2,752,181	237,582
Test sheet with 100 holes rotated (Figure 5b)	256 x 3 x 256	102,812	143,587	11,371
Rook (Figure 6)	31.75 x 31.75 x 53.15	93,930	5,573	575

3.2 Edge Matching

The sorting process in the ECC algorithm relies on comparing the edge pointer at the start and end of each list of connected vertices for a match with the current IN. This process can be further optimised using the fact that triangular meshes can only be matched edge to edge and vertex to vertex, therefore one triangle only shares edges with exactly three others, one on each edge. Consequently, once a match has been made for one intersection on an edge, it's matched triangle will be the same for the rest of that edge (Figure 7). The standard ECC algorithm was adjusted to account for this to reduce the number of sorting procedures required for each edge from *n*, total number of intersections of the edge to one.

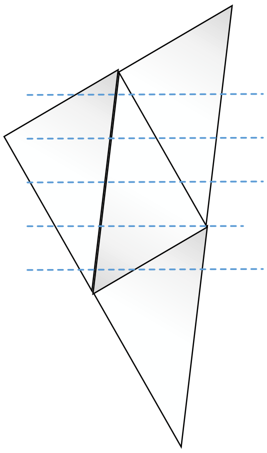


Figure 7: Example slice plane intersecting with matched triangles

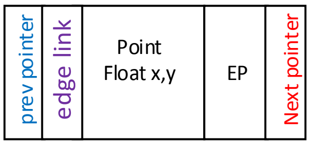


Figure 8: Modified IN to include the edge link

The IN data structure was modified to contain an edge link pointer (Figure 8), holding the memory location address which points to the ILL containing the IN of the subsequent intersection on the triangle. The procedure of using the edge link is described in the flowchart in Figure 9. The first intersection is inserted into the ILL using the method for the standard ECC algorithm and following this, the subsequent IN (IN$_i$) is inserted into the ILL under four cases depending on the outcome of the sorting of the previous IN (IN$_{prev}$):

1. If the IN$_{prev}$ is inserted into a new ILL, the current IN for consideration IN$_i$ will also be inserted into a new ILL, the address of the ILL containing IN$_i$ will be saved as IN$_{prev}$'s edge link pointer;
2. If IN$_{prev}$ is inserted at the back of a list, IN$_i$ will be inserted into the back of the list that is located at the edge pointer of the last IN in the list that IN$_{prev}$ connected to;
3. If IN$_{prev}$ is inserted at the front of a list, IN$_i$ will be inserted into the front of the list that is located at the edge pointer of the first IN in the list that IN$_{prev}$ connected to;
4. Finally, if IN$_{prev}$ connected two lists, IN$_i$ will connect the two lists contained in the edge pointers of the last IN in the ILL that IN$_{prev}$ follows and the first IN that IN$_{prev}$ precedes once connected.

In cases 1 to 3, the memory location address of the ILL that IN$_i$ has been inserted into is assigned as the edge link of the IN$_{prev}$. In case 4 it is unnecessary to assign the edge link, as IN$_{prev}$ is on the middle of an ILL and therefore will not be checked in future IN insertions. The impact of implementing the edge link pointer into the algorithm is that the number of intersections that undergo the checking process per triangle is reduced from *n* (the total number of intersections on the triangle) to one. The values recorded in counter temporarily implemented within each algorithm detailing the number of times the matching process of the standard ECC has undergone for each algorithm is given in Table 2.

Table 2: Reduction in the number of matching processes from ECC to EECC

Model	**Total number of sorts per part ECC**	**Average number of sorts per layer**	**Total number of sorts EECC**	**Percentage reduction from ECC to EECC**
Test sheet with 100 holes (Figure 5a)	1,562,144	52,071	53,555	2,916%
Test sheet with 100 holes rotated (Figure 5b)	2,380,123	898	53,555	44,446%
Rook (Figure 6)	3,949,700	7,452	81,435	4,850%

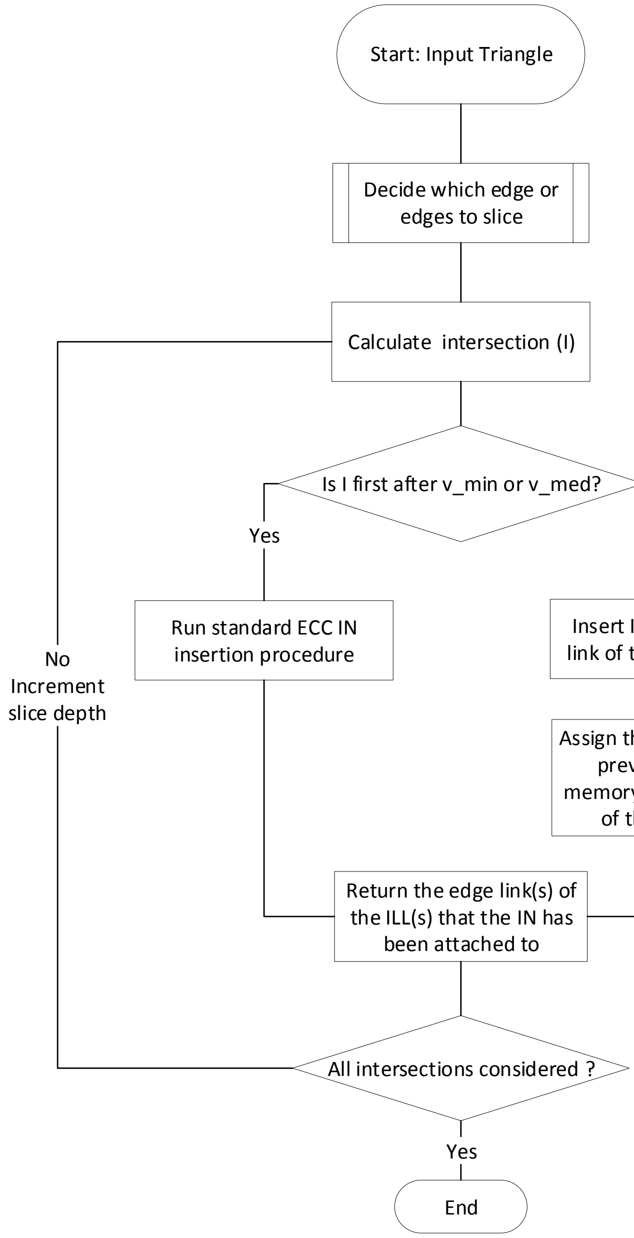


Figure 9: EECC algorithm flowchart

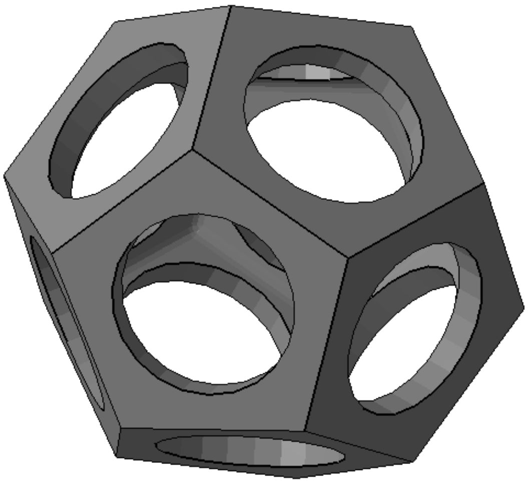


Figure 10: Dodecahedra model consisting of 2,074 triangles, 157.81 x 133.62 x 165.15mm

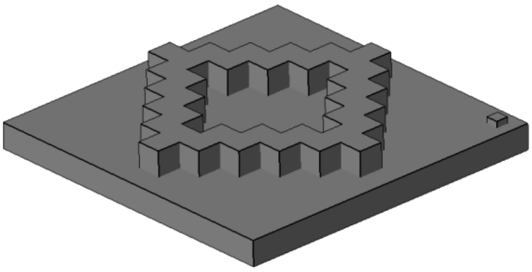


Figure 11: A calibration model consisting of 316 triangles, 165.1 x 165.1 x 25.4mm

3.3 Additional Modifications and edge matching

Transferring completed ILLs from the active sorting CLL to a separate CLL containing only completed lists, was expected to increase efficiency by reducing the number of sorts. In some cases, all the triangles in one connected multi-shell triangular mesh appear successively in the file; therefore, the contour or contours associated with that shell will be completed first, consequently, any further INs generated by the remaining shells will check the completed ILL on each search iteration. On initialisation of the algorithm, two versions of the CLL are created; the standard CLL where all sorting and IN insertions take place and a second complete CLL where ILLs are transferred by modifying the memory location pointers, when the IN is found to match in the same list twice.

Micro-optimisation in the order that case variables are assessed in the IF / ELSE-case loop provided a noticeable time saving. The order the case variables appear were restructured to ensure that the most likely case arises first. To test which case is the most likely, a series of integer values were created to count the number of times each case variable appeared. Table 3 shows the number of occurrences of each case recorded using integer counters when running the ECC algorithm with edge matching.

Table 3: EECC number of case occurrences per part

Model	No matching list	Insert at front of list	Insert at back of list	Link two lists	Requires Sorting
Test sheet with 100 holes (Figure 5a)	733,700	174	11,600	733,700	54,036
Test sheet with 100 holes rotated (Figure 5b)	377,964	256,360	25,639	800,658	72,589
Test sheet with 1225 holes (Figure 12)	17,904,600	174	497,350	17,904,600	1,288,736
Test sheet with 225 holes (Figure 15)	3,288,600	174	91,350	3,288,600	236,736
Test sheet with 484 holes (Figure 13)	7,074,144	174	19,504	7,074,144	509,204
Test sheet with 729 holes (Figure 14)	10,655,064	174	295,974	10,655,064	766,944
Dodecahedra (Figure 10)	4,850	7,810	1,386	4,693	567
Calibration model (Figure 11)	69,540	123,906	51,421	68,144	81,459
Rook (Figure 6)	30,379	83,907	82,258	26,514	9,077
Figure head (Figure 16)	159,372	194,935	202,963	15,906	511,004

Table 4 shows that the most dominant case is largely dependent on the topology of the triangles in the mesh, revealed by a comparison of the rotated test sheets in Figure 5 presenting differing case occurrences despite having triangles of identical geometry and connections. Table 5 contains the average likelihood of a case occurring for all models considered in this paper. It was determined that the order of the automatic insertion processes in the IF / ELSE loop would follow the most probable to the least probable occurrence outlined in Table 4.

Table 4: Average percentage case occurrences of the EECC sorting result on 50 test parts

No matching list (%)	Insert at front of list (%)	Insert at back of list (%)	Links two list (%)
26.91	29.89	20.79	22.40

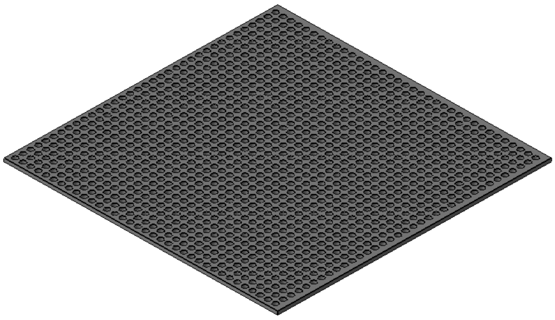


Figure 12: Test sheet containing 1,225 holes, 2,513,751 triangles of dimension 250x250x3mm

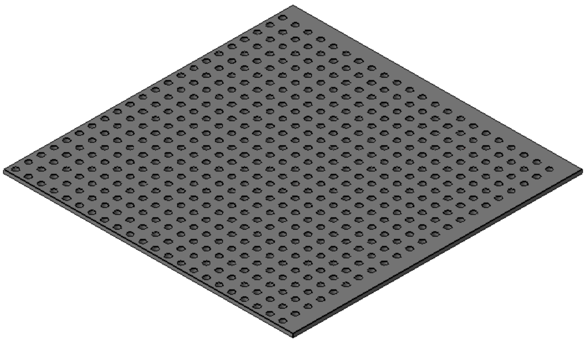


Figure 13: Test sheet containing 484 holes, 993,180 triangles of dimension 250x250x3mm

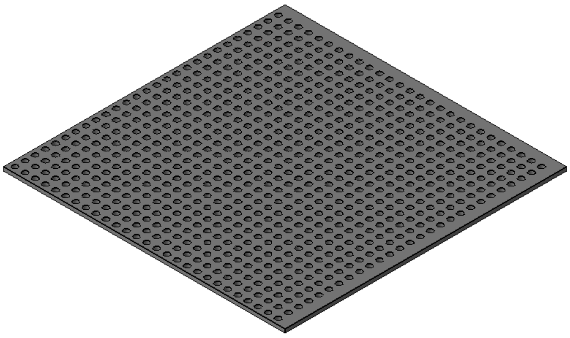


Figure 14: Test sheet containing 729 holes, 1,495,920 triangles of dimension 250x250x3mm

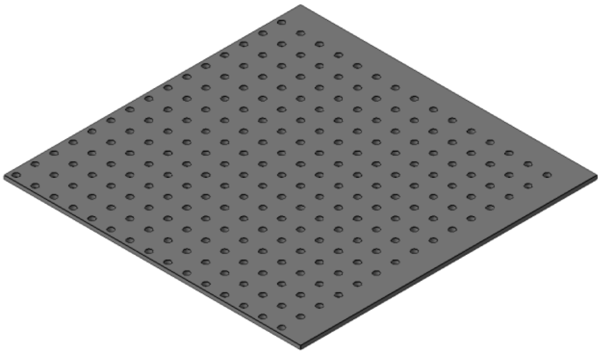


Figure 15: Test sheet containing 225 holes, 461,712 triangles of dimension 250x250x3mm

4 Results, Analysis and Discussion

The modifications to the ECC algorithm create the Enhanced Efficient Contour Construction (EECC) algorithm, to assess the successfulness of this method, several parts were sliced using the standard ECC algorithm, the ECC algorithm with edge matching, and the completed EECC algorithm, the results are shown in Table 5. The most impactful improvement is shown in the identified worst-case parts (Figures 12 to 15). In the largest worst-case model that could be sliced using the ECC algorithm (Figure 15), the EECC algorithm is 9,400% faster than the standard ECC algorithm. Of the parts tested, only models containing under 1,000 triangles witnessed a significant percentage increase in comparison to the original slicing time of 180% for Figure 10. This is an acceptable increase due to the imperceptible slicing times both with the ECC and EECC algorithm and can be explained due to the implementation of edge point testing taking more time than its saves; only a negligible decrease is seen in Figure 11 for the same reason.

Table 5: Enhanced ECC algorithm efficiency test results, all parts were sliced at 0.1mm slice thickness on a 64 bit system

Part name	**#Triangles**	**#Layers**	**Time conventional algorithm (s)**	**Time ECC (s)**	**Time ECC+ edge matching (s)**	**Time enhanced ECC (s)**
Test sheet with 100 holes (Figure 5a)	102,812	30	2752.181	237.582	10.029	8.241
Test sheet with 1225 holes (Figure 12)	2,513,712	30	-[1]	-[1]	1025.81	302.8
Test sheet with 225 holes (Figure 15)	461,712	30	-[1]	4775.382	70.884	50.3
Test sheet with 484 holes (Figure 13)	993,180	30	-[1]	-[1]	202.745	111.3
Test sheet with 729 holes (Figure 14)	1,495,920	30	-[1]	-[1]	380.788	169.1
Dodecahedra (Figure 10)	2,074	1,653	2.582	0.41	0.685	0.654
Calibration model (Figure 11)	316	255	1.573	0.031	0.066	0.058
Rook (Figure 6)	93,930	533	34.81	2.855	2.452	2.29
Figure head (Figure 16)	467,882	814	788.698	65.695	8.091	7.59
Lattice sole (Figure 17)	862,014	545	-[1]	-[1]	186.351	95.876

For Figures 11, 12 and 13, which represent the largest of the files tested using the standard ECC algorithm, underwent the slicing process for over 4 hours, but never reached completion due to the program being terminated after this time, as it was unacceptably long. This indicated that these parts would have seen even larger improvements than those witnessed in Figure 14, if completion was indeed possible

Figure 16: Figure head containing 467,882 triangles, dimensions of 141.61 x 111.27 x 81.29mm

---

[1] Slice time was in excess of four hours, process was terminated

Table 6 offers a comparison of open source slicers with the enhanced ECC algorithm, slicing was precisely timed by downloading the open source software and including timing modifications. The EECC algorithm was at least twice as fast for all instances.

Table 6: Comparison of the enhanced ECC algorithm with Slic3r and Cura, sliced at 0.1mm slice thickness with 0% infill

File Name	Ultimaker Cura(1) (s)	Slic3r(2) (s)	Enhanced ECC (s)
Dodecahedra (Figure 10)	7.674	9.125	0.654
Test Sheet with 225 holes (Figure 15)	128.360	-[2]	50.3
Rook (Figure 6)	5.341	6.31	2.29

4.1 Space and Time Complexity

The standard ECC sort procedure can be defined under three cases: worst case, best case and average case, if there are *k* number of lists in the CLL, m intersections per triangle and n triangles in the model – the complexity of the sorting algorithm is detailed in Table 7.

Table 7: Time complexity of the standard ECC sorting algorithm

Case	**Number of checks per intersection**	**Number of checks per triangle**	**Number of checks per model**
Best	$O(1)$	$O(m)$	$O(mn)$
Worst	$O(k)$	$O(km)$	$O(kmn)$
Average	$O(k/2)$	$O(km/2)$	$O(kmn/2)$

The introduction of the Enhanced ECC algorithm reduces the number of sorts per triangle from m intersections on the triangle to 2 in all cases and therefore the time complexity become $O(2n)$, $O(2kn)$ and $O(kn)$ for best, worst and the average case respectively. This demonstrates that the improvements to the ECC algorithm has the greatest impact on the worst case triangular meshes, and the least on the best case. The worst case sort procedure can be differentiated from previously identified worst case models where the k value would be very large, up to 67% of the total number of triangles n, when compared to a best case model where k would be less than 1% of the total number of triangles.

There was a slight increase in space complexity in the enhanced ECC algorithm in comparison to the standard ECC algorithm due to the implementation of the edge link pointer, where each pointer is eight bytes on a 64-bit system. The total space requirement for one intersection is four bytes each for the X and Y coordinate of the intersection and five pointers, two edge pointers, one edge link pointer and the two pointers which link the contour together, which is a total of 48 bytes per intersection, an increase of eight bytes or 16.67% over the standard ECC algorithm. As there are m intersections per triangle and n triangles in the model, the total RAM requirement can be defined as 48nm bytes. This slight increase in space complexity can be justified by the improvements in efficiency.

---

[2] Slicing could not complete without program terminating

4.2 Industrial Context

Lattice structures have been identified as offering significant advantages over solid infill products, design dependent they can offer the same or better material properties e.g. tensile and compressive strength at a considerably reduced part weight and volume. These types of parts have seen significant advantages in areas where a high strength to weight ratio is desirable, examples include aerospace and sports performance products. Lattice structure models can often be categorised as worst-case models, especially when the lattice is in one layer running from top to bottom in the direction of construction.

One industrial example of lattice structures in AM is 3D printed shoes [31], Figure 17 shows the Adidas Alphaedge 4D shoes currently available on the mass market, featuring a lattice structure on the sole of the shoe. Increasingly these shoes are manufactured custom to a scan of the wearers foot, meaning that each CAD model is different and will need to be sliced individually, resulting in overall very lengthy slice times. Figure 18 shows a model of the sole of shoe intended that is intended for production using additive manufacturing. This part can be considered both worst case, with an average of 24% triangles intersecting on each layer and a large *.stl file. The results in Table 5 demonstrate enhanced ECC algorithm offers significant advantage on this part that would be manufactured in an industrial application. The part shows an improvement of over 100% on the standard ECC algorithm and an improvement of at least 15,200% over the traditional end to end line sort algorithm.

Figure 17: Adidas AlphaEdge 4D [31]

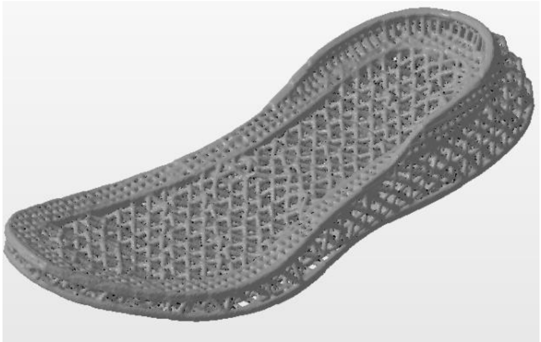


Figure 18: Lattice sole of AM manufactured shoe, containing 862,014 triangles, dimensions of 324 x 125 x 54mm

5 Conclusion

The objective of this research was to generate a slicing algorithm for AM that is capable of efficiently slicing worst case geometric parts, defined as triangular mesh models where a high percentage of the parts triangles intersect on each layer. An adaption of the ECC algorithm, including reduction in the number of sorts for each triangle, and micro-optimisations through structuring, formed the enhanced ECC algorithm. Efficiency tests were conducted on a set of *.STL files (however, any other triangular mesh files could be used in the algorithm) and found a maximum improvement of 9,400% on the largest worst-case file. It was also found that *.STL files that were previously too time inefficient to complete slicing using the standard ECC algorithm took less than 300s to slice.

The enhanced ECC algorithm addresses the failings of the other algorithms to slice very large worst-case parts, which are becoming more prevalent in the AM sector [29] in reproduction of scanned real-world objects [30] or highly detailed, large scale AM components. Improvements to the slicing process will have to evolve as the models grow in complexity and size; whilst the Enhanced ECC algorithm may be able to slice all parts efficiently now, further developments will be necessary in the future.

Declarations

Ethical Approval: This study complies with the ethical standards set out by springer

Consent to Participate: Not applicable

Consent to Publish: Not applicable

Author Contributions: BK undertook the development of the algorithm presented in this paper, supervised by AR. Models for testing were supplied by GB

Funding: This study part funded by the Low Carbon in Lancashire Hub grant reference 19R16P01012 and Euriscus Ltd.

Competing interests: This research is sponsored by Euriscus Ltd of which Graham Bennett is the CTO

Availability of Data and materials: Not applicable.

References

[1] Gibson, I., Rosen, D. & Stucker, B. (2014) *Additive Manufacturing Technologies: 3D Printing, Rapid Prototyping, and Direct Digital Manufacturing.* 2nd ed.: Springer, New York.

[2] King, B., Bennett, G.R. and Rennie, A.E.W. (2017). Comparison of Galvanometer and Polygon Scanning Systems on Component Production Rates in Selective Laser Sintering, *Proceedings of the 15th Rapid Design, Prototyping & Manufacturing Conference (RDPM2017)*, Newcastle, UK, ISBN 978-1-5272-2153-6.

[3] Mohsen, A. (2017) The rise of 3-D printing: The advantages of additive manufacturing over traditional manufacturing. *Business Horizons* 30(5):677-688.

[4] Garden, J. (2016) Additive manufacturing technologies: state of the art and trends, *International Journal of Production Research,* 54(10):3118-3152

[5] Paul R & Anand S, “A new Steiner patch-based file format for Additive Manufacturing Processes”, Computer-Aided Design 63 (2015) 86-100.

[6] Cătălin I., Daniela I. & Alin S. (2010) From Cad Model To 3d Print Via “Stl” File Format. *Fiabilitate şi Durabilitate,* l 1(5)**:**73-80.

[7] Adnan, F.A., Romlay, F.R.M. & Shafiq, M. (2018) Real-time slicing algorithm for Stereolithography (STL) CAD model applied in additive manufacturing industry, IOP conference series Materials Science and Engineering, Vol. 342(1):012016.

[8] Engstrom, D., Porter, B., Pacios, M. & Bhaskaran, H. (2014) Additive nanomanufacturing - A review. *J. Mater. Res.,* , 29(17):1792-1816.

[9] Zha, W. & Anand, S. (2015) Geometric approaches to input file modification for part quality improvement in additive manufacturing. *Journal of Manufacturing Processes,* , 20 :465-477.

[10] Zhang, Z. & Joshi, S. (2015) An improved slicing algorithm with efficient contour construction using STL files. *The International Journal of Advanced Manufacturing Technology,* , 80(5-8): 1347.

[11] Brown, A.C. & De Beer, D. (2013) Development of a stereolithography (STL) slicing and G-code generation algorithm for an entry-level 3-D printer, *2013 AFRICON*, 1-5.

[12] Xu, H., Jing, W., Li, M. & Li, W (2016) A slicing model algorithm based on STL model for additive manufacturing processes. *IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)* : 1607-1610.

[13] Tian, R., Liu, S. & Zhang, Y. (2018) Research on fast grouping slice algorithm for STL model in rapid prototyping *J. Phys.: Conf. Ser*. 1074 012165

[14] Steuben, J., Iliopoulos, A., Michopoulos, J. Implicit slicing for functionally tailored additive manufacturing. *Computer Aided Design.* , 77: 107-119

[15] Hopcroft, J., Tarjan, R. (1973) Efficient algorithms for graph manipulation *Commun ACM,*16 (6) : 372-378,

[16] Hu, B., Jin, G. & Sun, L. (2018) A Novel Adaptive Slicing Method for Additive Manufacturing, *CSCWD*, Nanjing, 2018, : 218-223.

[17] Wang, W., Chao, H., Tong, J., Yang, Z., Tong, X., Li, H., Liu, X. & Liu, L. (2015) Saliency-Preserving Slicing Optimization for Effective 3D Printing. *Computer Graphics Forum* 34(6) :148-160.

[18] Pan, X., Chen, K., Zhang, Z., Chen, D. & Li, T. (2013) Adaptive slicing algorithm based on STL model, *Applied Mechanics and Materials* 288:241-245.

[19] Li, Q. & Xu, X. Y. (2015) Self-adaptive slicing algorithm for 3D printing of FGM components. *Materials Research Innovations* 19(S5): 635-641.

[20] Ding, D., Pan, Z., Cuiuri, D., Li, H., Larkin, N. & Van Duin, S. (2016) Automatic multi-direction slicing algorithms for wire based additive manufacturing. *Robotics and Computer Integrated Manufacturing,* 37**:**139-150.

[21] Singhal, S.K., Jain, P.K. & Pandey, P.M. (2008) Adaptive Slicing for SLS Prototyping. *Computer-Aided Design and Applications* 5(1-4) : 412-423.

[22] Pereira, S., Vaz, A. & Vicente, L. (2018) On the optimal object orientation in additive manufacturing. *The International Journal of Advanced Manufacturing Technology* 98(5) :1685-1694.

[23] Golmohammadi, A.H. & Khodaygan, S. (2019) A framework for multi-objective optimisation of 3D part-build orientation with a desired angular resolution in additive manufacturing processes. *Virtual and Physical Prototyping,* 14(1) :19-36.

[24] Yang, G., Liu, W., Wang, W. & Qin, L. (2010) Research on the rapid slicing algorithm based on STL topology construction. *Advanced Materials Research* 97-101 : 3397-3402.

[25] Eragubi, E. (2013) Slicing 3D Model in STL Format and Laser Path Generation. *International Journal of Innovation, Management and Technology* 4(4) :410-413.

[26] Zhao, G., Ma, G., Feng, J. & Xiao, W. (2018) Nonplanar slicing and path generation methods for robotic additive manufacturing.(Report). *The International Journal of Advanced Manufacturing Technology,* 96(9-12) : 3149.

[27] Luo, N. & Wang, Q. (2016) Fast slicing orientation determining and optimizing algorithm for least ,umetric error in rapid prototyping, *The International Journal of Advanced Manufacturing Technology* 83(5):1297-1313

[28] Zhang, Y., De Backer, W., Harik, R. & Bernard, A. (2016) Build Orientation Determination for Multi-material Deposition Additive Manufacturing with Continuous Fibers, *Procedia CIRP* 50, :414-419.

[29] Barnett, E. & Gosselin, C. (2015) Large-scale 3D printing with a cable-suspended robot. *Additive Manufacturing* 7(C) :27-44.

[30] Tóth, T. & Živčák, J. (2014) A Comparison of the Outputs of 3D Scanners, *Procedia Engineering* 69, :393-401.

[31] Cătălin A. , Zapciu A. , & Popescu D. (2019). 3D-Printed shoe last for bespoke shoe manufacturing. *MATEC Web of Conferences, 290*, 4001.

[32] Perry A. (2018) 3D-printed apparel and 3D-printer: exploring advantages, concerns, and purchases, *International Journal of fashion design and innovation*, 11(1), 95-103