Contour generation algorithm for Projection Mask Stereolithography 3D printing process

F A Adnan[1], **F R M Romlay**[1*]

[1]Faculty of Mechanical Engineering, Universiti Malaysia Pahang, 26600 Pekan, Pahang, Malaysia

[*]Corresponding author: fadhlur@ump.edu.my

Abstract. Projection mask stereolithography is a latest technology in 3D printing industry. It relies on UV projection to cure the photocurable resin layer-by-layer to build the solid 3D object. The projections are generated from tessellated STL CAD model using contour generation algorithm that generates 2D contours as the projection mask. The computational of the algorithm consumes a lot of memory for high-resolution printing. Low-resolution causes the stair-case effect to visibly appear on the printed model. Using real-time contour generation algorithm reduces the memory consumption and retains the high-resolution printing quality and accuracy. Pixel line mapping algorithm is implemented on the generated line segments to scale the line segments with respect to the display resolution of the projection device. Contour loop algorithm connects each of these pixel line segments into multiple closed-loop contours. The algorithms are implemented on an Alien STL model having 150350 facets. It is found that the pixel mapping algorithm correctly scales the model while retaining the aspect ratio of the model. The result of contour loop algorithm shows that the mean contour loop execution time is 0.87 milliseconds which is relevant for real-time application.

1. Introduction

Projection mask stereolithography is one of recent 3D printing technology. It utilizes ultraviolet (UV) projection through an oxygen permeable window under a bed of photocurable resin. The UV light passing through the window solidifies the resin to form a solid layer that adheres to the build platform. The build platform is then elevated to introduce an uncured resin for the printer to start building the next layer [1]. This process is repeated with different layer of contour projection depending on the current build height of the STL model until the 3D model is completely printed [1-4]. Projection mask 3D printing method is known to surpass any other 3D printing technology in terms of accuracy due to its uniform layer curing process [4, 5]. This continuous curing process causes the printed part to become monolithic. This highly improves its mechanical properties.

Past researchers have addressed multiple issues regarding the computational cost of the contour generation algorithm [7, 8] which generates the digital contour mask used in projection-based 3D printer. For high-resolution printing, the thickness between each slices of contours must be very small [9]. This often consumes a lot of memory to store the generated contours [10]. However, if the slicing thickness is too thick (low resolution), it will cause stair-case effect to visibly appear on the printed object. Some researchers have proposed adaptive slicing algorithm to overcome the issue of memory consumption and stair-case effect [11]. The adaptive slicing algorithm varies the slicing thickness by limiting the computed cusp height according to the respective STL model [12]. This method significantly reduces the memory consumption compared to the traditional uniform slicing technique (fixed slicing thickness).

Another issue commonly addressed by past researchers are the computational time of the contour generation algorithms. Extensive works have been done to improve the computational time. Among the methods proposed, is the use of Octree data structure which allows the program to rapidly access the facet data stored in the memory for contour generation processes [10, 11]. This research shows the importance of using the right data structure to optimize the computational time of the algorithm. Storing the facets in a linear data structure such as an array will has the worst case of O(n) which is time consuming since the algorithm has to search through each element in the array. In the work of Minetto, they proposed an optimal slicing algorithm and implemented for both adaptive and uniform slicing schemes. Linear data structure is still being used but the facets are sorted in multiple list of facets with varying slicing height. The proposed slicing algorithm has the time complexity of O(n log k + k + m). Time measurements were presented and compared with other similar research. It was found that their proposed algorithm performs much better than the rest [15]. However, this work is considered as general algorithm in computational graphic field of study. It was constructed without any specific applications.

This stems our work on contour generation algorithm specifically for projection-based 3D printing applications. Our work focuses on real-time slicing where only single layer of contours is instantly generated by constantly monitoring the current printing height to solve the memory and accuracy issues. This paper is the extension of our previous work on slicing algorithm. In our previous work, the fundamental of slicing algorithm was presented [16]. The implementation of the slicing algorithm on STL facets generate multiple arbitrary line segments [16, 17]. These line segments are not programmatically connected. Thus, in this paper, an algorithm for connecting each of the line segments to form contours are presented. Other than that, with projection-based 3D printing application in mind, this paper also proposes pixel line mapping algorithm to scale the line segment with respect to display resolution of the output projection device to improve the performance of the algorithm.

2. Experimental Setup

The code is written in C++11 programming language in Qt Creator 4.6.0 with Community license and compiled using MinGW C++ Compiler. Several built-in libraries are used in C++11 such as <*math*> for mathematical operations, <*vector*> for dynamic data structure, and <*chrono*> for high-resolution time measurement. The program is executed on Intel i3-2350M 2.30 GHz (2 Cores/4 Threads) 6GB RAM laptop which runs on Windows 7 Home 64-bit.

3. Methodology

3.1. Pixel Line Mapping Algorithm

Regardless how high the number precision of the generated line coordinates, in the end it all will be scaled back to round integer pixel coordinate of the display device [5]. Floating numbers are prone to have truncation errors which lead to disconnection between each generated line segments [14, 15]. Furthermore, floating numbers are much slower to process compared to rounded integers. Thus, pixel line mapping algorithm scales all the line segments into pixel coordinate system with respect to the resolution of the projection device to improve the contour loop algorithm performance. This can be achieved by first comparing the aspect ratio of the device display resolution and the object resolution by using equation (1) and (2) below.

$$\mathrm{AR}_{\mathrm{device}} = \frac{\mathrm{width}}{\mathrm{height}} \tag{1}$$

$$\mathrm{AR}_{\mathrm{object}} = \frac{\mathrm{x}_{\mathrm{max}} - \mathrm{x}_{\mathrm{min}}}{\mathrm{y}_{\mathrm{max}} - \mathrm{y}_{\mathrm{min}}} \tag{2}$$

where $\mathrm{AR}_{\mathrm{device}}$ is the aspect ratio of the display width divided by its height. $\mathrm{AR}_{\mathrm{object}}$ is the width and height ratio of the object. Next, a piecewise function, R is introduced to handle both case when $\mathrm{AR}_{\mathrm{device}}$ is more than $\mathrm{AR}_{\mathrm{object}}$ or vice versa in equation (3).

$$R = \begin{cases} \text{width} - 1; AR_{object} \geq AR_{device} \\ \text{height} - 1; AR_{object} < AR_{device} \end{cases} \quad (3)$$

The piecewise function R defines whether the object should be scaled to fit the width or the height of the projection device. This is to prevent the object from exceeding the display resolution of the projection device. Next, new piecewise function V and W to act as modifiers for pixel coordinate in x-axis and y-axis respectively.

$$V = \begin{cases} 1 \quad ; AR_{object} \geq AR_{device} \\ AR_{object}; AR_{object} < AR_{device} \end{cases} \quad (4)$$

$$W = \begin{cases} AR_{object}^{-1}; AR_{object} \geq AR_{device} \\ 1 \quad ; AR_{object} < AR_{device} \end{cases} \quad (5)$$

Using linear interpolation equation combined with all 3 piecewise functions in (3), (4), and (5), the line pixel mapping function can be written as:

$$x_{pixel} = \frac{x - x_{min}}{x_{max} - x_{min}} \cdot R \cdot V \quad (6)$$

$$y_{pixel} = \frac{y - y_{min}}{y_{max} - y_{min}} \cdot R \cdot W \quad (7)$$

Equation (6) and (7) convert the floating-point coordinates into pixel coordinate of the projection device. In the equations, x and y are the point to be converted. The x_{min} and y_{min} are the lowest point of the object with respect to its axis. The x_{max} and y_{max} are the highest point of the object with respect to its axis.

3.2. Contour Loop Algorithm

This algorithm is the algorithm which connects all the line pixel segments generated in section 3.1 into multiple closed-loop contours. Basically, the algorithm is a simple head-to-tail search algorithm that look for pairs of lines by comparing its point coordinates. Each line segment has two distinct point which are denoted as P_o and P_f. During slicing process, there is a probability that the generated lines are inverted due to the facet facing inward. Thus, the contour loop algorithm should be able to detect when the line is inverted and then flipped the line back to the right orientation. Below is the contour loop algorithm table.

Algorithm: Contour Loop

```
1: initialize id ← 0, P_init ← S[0].P_o
2: for i = 0 to S.length − 1 do
3:     P_find ← S[i].P_f
4:     if Compare(P_find, P_init) then
5:         id = id + 1
6:         P_init ← S[i + 1].P_o
7:     else
8:         found ← Find(i + 1, P_find)
9:         if found ≠ −1 then
10:            isInvert ← Compare(P_find, S[found].P_f)
11:            Swap(S[found], S[i + 1], isInvert)
12:        end if
13:    end if
14:    S[i + 1].id ← id
15: end for
```

In Step 1, the algorithm initializes the variable id as zero. The id variable indicates which contour group does a line belongs to. Assuming that the line begins from P_o and ends with P_f, the algorithm assigns the first item P_o from the list of line segment S to a temporary variable P_{init} to acts as the head of the contour loop. Step 3 assigns the first element P_f from list S to P_{find} as the current tail of the contour loop. In Step 8, the algorithm uses P_{find} and $i + 1$ iteration for the Find function to search the next line in list S which has the same point coordinates as P_{find}. The $i + 1$ iteration is an offset of where the Find function should start searching from the list S. If the function found the matched line segment, the function returns the index of the found point in variable found. Else, it will return -1 to indicate that no match is found. In Step 10, a Boolean Compare function checks whether P_{find} is similar to P_f of the found line. If it is similar, this means that the found line is inverted, thus returns true. Step 11 swaps the found line with next line segment from the list. If the Boolean variable isInvert is true, then in the Swap function, the found line P_o will be swapped with its P_f to correct the line orientation. Now the next point in the list S is the newly found line. Step 14 assigns the id to the newly found line to indicate which contour group the line belongs to. The iteration repeats by reassigning new P_{find} with the newly found line P_f that was previously placed in the next sequence of list S. Step 4 compares the P_{find} and the P_{init}. If they are the same, it means that the contour tail has met its head, thus forms a closed loop. Then, the variable id is incremented to indicate different contour group in Step 5. Step 6 reassigns new head for the new contour group. The cycle repeats until the last element of the list S.

4. Results and discussion

Figure 1 below shows the result of contour generation algorithms. The algorithms are implemented on Alien STL model with 150350 facet counts and sliced with 300 slicing planes with uniform thickness. As seen in Figure 1, each layer is stacked on top of each other to reconstruct the 3D model. Contour lines can be observed appearing on the model. The contour generation algorithm perfectly sliced the STL model into 2D contour layers for the application of projection-based 3D printing.

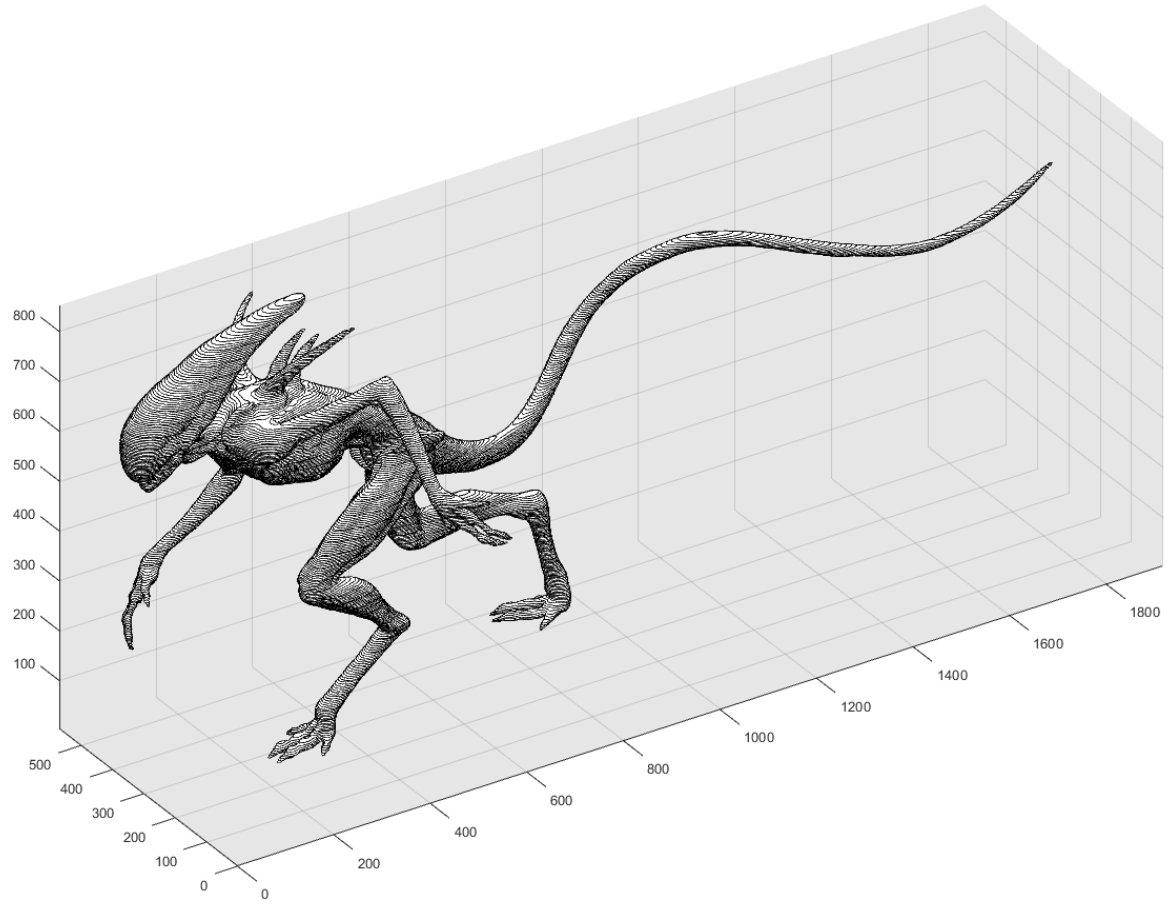

Figure 1. Generated contours stacked with respective slicing height to form 3D model.

The performance of the contour loop algorithm discussed in section 3.2 is plotted in the graph below. The first graph (a) shows the computational time taken by the contour loop algorithm to process the layer with respect to each slicing height. Second graph (b) shows the number of contour loop counted at each slicing height. Third graph (c) plots the number of facets which intersect with the slicing plane at respective slicing height.

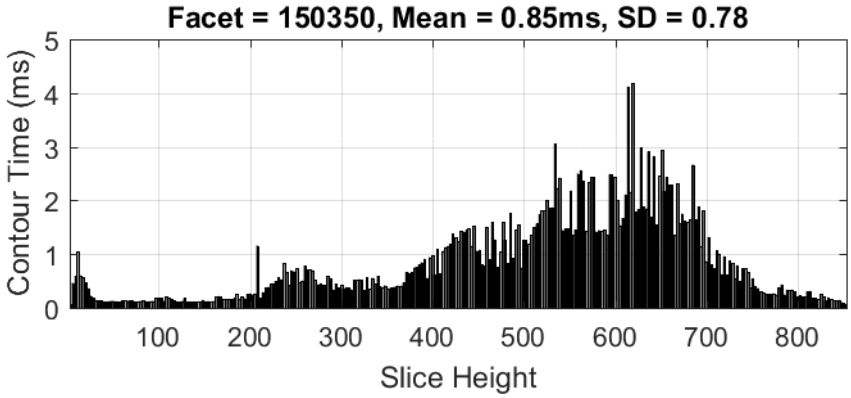

Figure 2a. Contour time result.

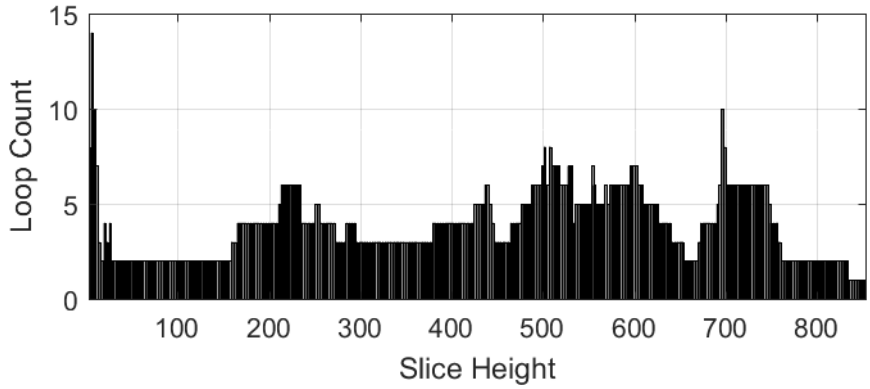

Figure 2b. Loop counts.

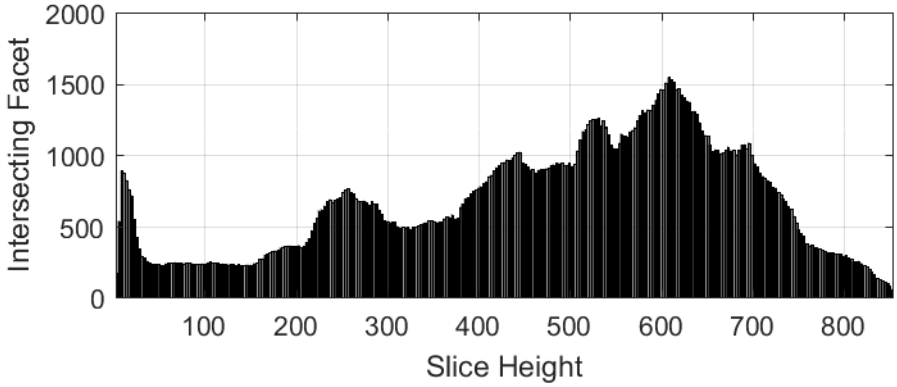

Figure 2c. Number of intersecting facets.

Graph (a) and (c) in Figure 2 can be observed to have similar patterns. This indicates a strong correlation between the contour time and the number of intersecting facets. Hence, a Normalized Correlation is performed to further supports the claim. The result is tabulated in Table 1 below.

Table 1. Result of Normalized Correlation.

Normalized Correlation	
Contour time vs. Loop count	0.7974
Contour time vs. Intersecting facet	0.9377

Table 1 proves strong correlation between the contour time and the number of intersecting facets at each slicing height. It can be said that the more the number of facets which intersect with the slicing plane, the higher the computational time it will takes to perform the contour loop algorithm.

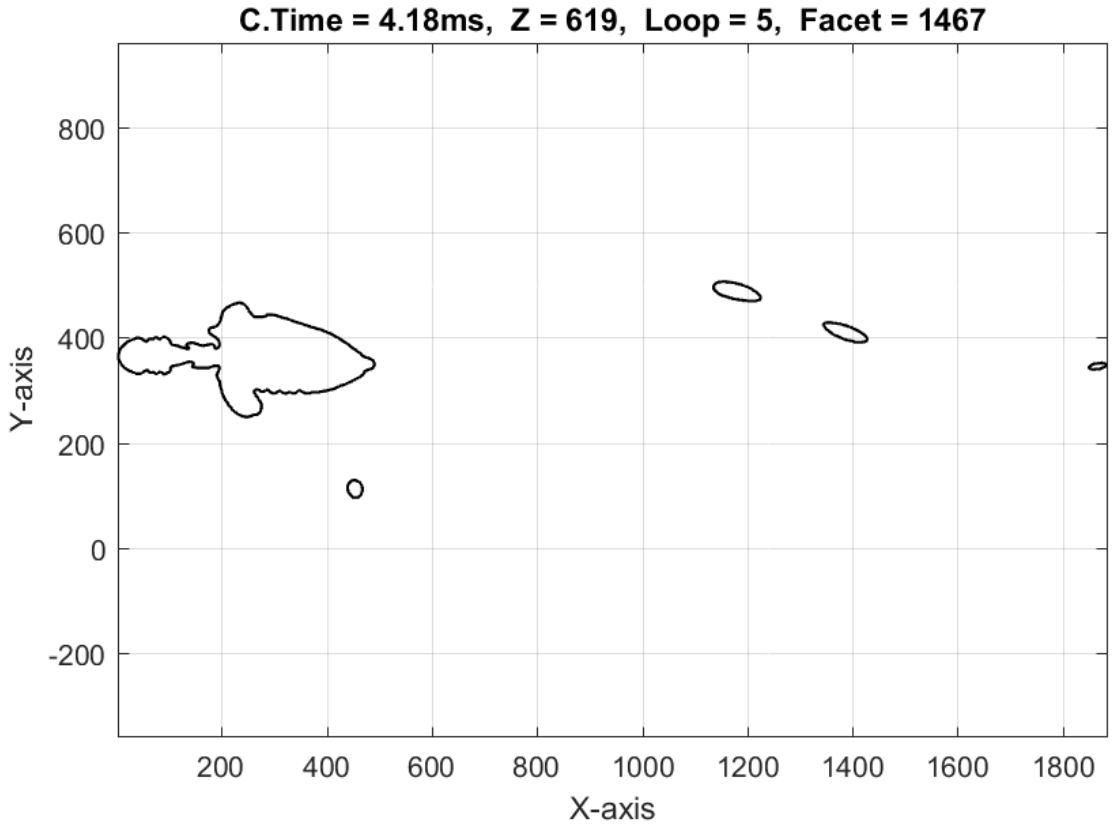

Figure 3. Contours with the highest computational time.

Figure 3 shows the layer with highest computational time of the contour loop algorithm which is located at height Z=619. There are 5 contour loops can be observed. The Loop at the top also shows 5 loops are generated. This proves that the contour loop algorithm correctly groups the contours into 5 contour groups. 1920 x 1080 resolution is set as the display resolution for the algorithm implementation. The contours in Figure 3 is seen to be within the specified resolution. This also proves that the line pixel mapping algorithm works. From the figure, this is considered as fit-to-width case where AR_{object} is more or equal to AR_{device} because the contours are mapped to x-axis starting from 0 to 1920 while retaining the object aspect ratio in y-axis. Although the contour does not reach 1920 in this layer, there are different layer at another slicing height that reached full x-axis scale of 1920.

5. Conclusion

The algorithm is implemented using an Alien STL model with facet counts of 150350. Display resolution is set to be 1920 x 1080 for the test. It is found that the line pixel mapping algorithm fulfilled its purpose which is to scale the generated line segments with respect to display resolution of the projection device. By using contour loop algorithm on the line pixel generated, the algorithm manages to group multiple closed-loop contour. Based on the Normalized Correlation, a strong correlation between the contour time and the number of facets which intersect with the slicing plane. The more the intersecting facet, the longer computational time for the contour loop algorithm. With the pixel line mapping algorithm, the contour loop generation algorithm is able to perform much faster with the measured time having relatively low mean computational time.

Acknowledgments

This research is funded by Universiti Malaysia Pahang (UMP) under grants RDU160387. The authors would like to express many thanks to UMP for supporting this research.

References

[1] Jing Hu, "Study On STL-Based Slicing Process For 3D Printing," *Solid Free. Fabr.*, pp. 885–895, 2017.

[2] J. R. Tumbleston *et al.*, "Continuous liquid interface production of 3D objects," *Science (80-.).*, vol. 347, no. 6228, pp. 1349–1352, 2015.

[3] Y. Pan, C. Zhou, and Y. Chen, "A Fast Mask Projection Stereolithography Process for Fabricating Digital Models in Minutes," *J. Manuf. Sci. Eng.*, vol. 134, no. 5, p. 051011, 2012.

[4] X. Zhao, "Process planning for thick-film mask projection micro stereolithography," 2009.

[5] H. Ye, C. Zhou, and W. Xu, "Image-Based Slicing and Tool Path Planning for Hybrid Stereolithography Additive Manufacturing," *J. Manuf. Sci. Eng.*, vol. 139, no. 7, p. 071006, 2017.

[6] C. Sun, N. Fang, D. M. Wu, and X. Zhang, "Projection micro-stereolithography using digital micro-mirror dynamic mask," *Sensors Actuators, A Phys.*, vol. 121, no. 1, pp. 113–120, 2005.

[7] Z. Zhang and S. Joshi, "An improved slicing algorithm with efficient contour construction using STL files," *Int. J. Adv. Manuf. Technol.*, vol. 80, no. 5–8, pp. 1347–1362, 2015.

[8] G. Zhao, G. Ma, J. Feng, and W. Xiao, "Nonplanar slicing and path generation methods for robotic additive manufacturing," *Int. J. Adv. Manuf. Technol.*, vol. 96, no. 9–12, pp. 3149–3159, 2018.

[9] A. W. A. Baqer and T. F. Abbas, "Direct slicing techniques of 3D surface model in layers manufacturing," *Int. J. Energy Environ.*, vol. 9, no. 4, pp. 407–414, 2018.

[10] M. Alexa, K. Hildebrand, and S. Lefebvre, "Optimal Discrete Slicing," *ACM Trans. Graph.*, vol. 36, no. 1, pp. 1–16, 2017.

[11] P. M. Pandey, N. V. Reddy, and S. G. Dhande, "Real time adaptive slicing for fused deposition modelling," *Int. J. Mach. Tools Manuf.*, vol. 43, no. 1, pp. 61–71, 2003.

[12] R. M. M. H. Gregori, N. Volpato, R. Minetto, and M. V. G. Da Silva, "Slicing Triangle Meshes: An Asymptotically Optimal Algorithm," *2014 14th Int. Conf. Comput. Sci. Its Appl.*, pp. 252–255, 2014.

[13] N. Siraskar, R. Paul, and S. Anand, "Adaptive Slicing in Additive Manufacturing Process Using a Modified Boundary Octree Data Structure," *J. Manuf. Sci. Eng.*, vol. 137, no. 1, p. 011007, 2015.

[14] H.-T. T. Wong, Y. Huang, S.-C. Tsang, and M.-L. Lam, "Real-time model slicing in arbitrary direction using octree," *ACM SIGGRAPH 2017 Posters - SIGGRAPH '17*, pp. 1–2, 2017.

[15] R. Minetto, N. Volpato, J. Stolfi, R. M. M. H. Gregori, and M. V. G. da Silva, "An optimal algorithm for 3D triangle mesh slicing," *CAD Comput. Aided Des.*, vol. 92, pp. 1–10, 2017.

[16] F. A. Adnan, F. R. M. Romlay, and M. Shafiq, "Real-time slicing algorithm for Stereolithography (STL) CAD model applied in additive manufacturing industry," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 342, no. 1, 2018.

[17] X. Pan, K. Chen, and D. Chen, "Development of rapid prototyping slicing software based on STL model," *Proc. 2014 IEEE 18th Int. Conf. Comput. Support. Coop. Work Des. CSCWD 2014*, no. 51175395, pp. 191–195, 2014.

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

[19] M. Szilvśi-Nagy and G. Mátyási, "Analysis of STL files," *Math. Comput. Model.*, vol. 38, no. 7–9, pp. 945–960, 2003.

[20] T. Wu and E. H. M. Cheung, "Enhanced STL," *Int. J. Adv. Manuf. Technol.*, vol. 29, no. 11–12, pp. 1143–1150, 2006.