Escaping Tree-Support (ET-Sup): Minimizing Contact Points for Tree-like Support Structures in Additive Manufacturing

Tsz-Ho Kwok

Department of Mechanical, Industrial and Aerospace Engineering, Concordia University, Montreal, Canada

Abstract

Purpose: Support structures are often needed in additive manufacturing (AM) to print overhangs. However, they are the extra materials that must be removed afterwards. When the supports have many contacts to the model or are even enclosed inside some concavities, removing them is very challenging and has a risk of damaging the part. Therefore, the purpose of this paper is to develop a new type of tree-support, named escaping tree-support (ET-Sup), which tries to build all the supports onto the build plate in order to minimize the number of contact points.
Methodology: The methodology is to first classify the support points into three categories: clear, obstructed, and enclosed. A clear point has nothing between it and the build plate; an obstructed point is not clear, but there exists a path for it to reach the build plate; and an enclosed point has no way to reach the build plate. With this classification, the path for the obstructed points to come clear can be found through linking them to the clear points. All the operations are performed efficiently with the help of a ray representation.
Findings: The method is tested on different overhang features, including a lattice ball and a mushroom shape with a concave cap. All the supports generated for the examples can find their way to the build plate, which looks like they are escaping from the model. The computation time is around one second for these cases.
Originality: This is the first time truly realizing this 'escaping' property in the generation of tree-like support structures. With this ET-Sup, it is expected that the AM industries can reduce the manufacturing lead time and save much labor work in post-processing.

Keywords: Support structure, Additive manufacturing, Tree-like support, Removability

1. Introduction

Additive manufacturing (AM), or three-dimensional (3D) printing, produces parts by accumulating material layer-by-layer. Despite its capability in building very complex geometry, it is limited by gravity and cannot add material in the air. When a model has overhangs (disconnected regions at a layer) or bridges (cantilever beams) that are not supported by anything below, the layered process leads inevitably to the need of support structures. Support structures have been considered a necessary evil in AM. On the one hand, they serve the purposes of part balancing and thermal dissipation besides ensuring the printability. On the other hand, since they are not part of the model, additional work is needed to remove them, which increases both material and labor costs, and they can damage the model. Therefore, many try to minimize the use of support structures by reorientation [1] or altering the geometry [2]. However, they are often a must for most real-world models, especially if they are complicated. When a printer can fabricate multiple materials, an excellent option is to use soluble support (e.g., polyvinyl alcohol and polystyrene), which can be dissolved away cleanly without worries [3]. Unfortunately, there are techniques only able to fabricate one material, for example, stereolithography (SLA), digital light processing (DLP), single-nozzle fused filament fabrication (FFF), electron-beam melting (EBM) [4], and selective laser melting (SLM) [5]. In these cases, the support structures must be printed with the same material, and they need to be broken away after the print.

A common type of the break-away support structures is the linear or lattice supports [8], which prop the entirety of the overhangs using a set of vertical pillars (see Fig. 1a). This kind of support is reliable, but they are

Email address: tszho.kwok@concordia.ca (Tsz-Ho Kwok)

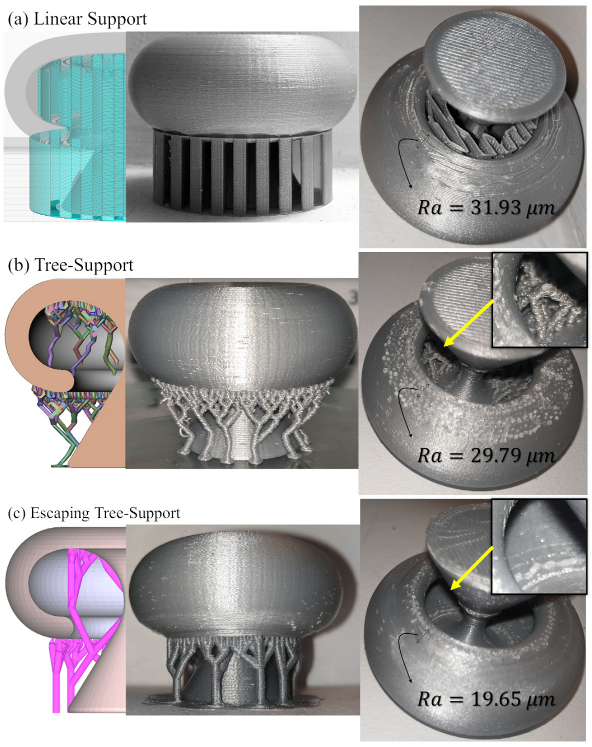

Figure 1: The support structures generated for a mushroom model: (a) linear support by Cura [6], (b) tree-like support by Meshmixer [7], and (c) escaping tree-support by the present method.

notoriously harder to remove and more likely to damage the model. Another type of supports is the tree-like supports [9], which grow branches to touch the overhangs (see Fig. 1b). Tree supports use less material and have less contacts with the overhangs, so that they are easier to remove and cause less damage. This is why the tree-like supports have drawn much attention in recent years. However, most existing works (will be reviewed) focus on reducing the manufacturing time and the material waste. Surprisingly, after having years of experience collaborating with 3D printing industries, it is found that they actually do not care that much of the added material cost. The reason behind is that the manufacturing cost of a part includes all the processes needed to produce the part. That is, it is not only determined by the print time and the material cost, but also the post-processing like support removal. The lower cost of one may be overwhelmed by the higher costs of other operations. For example, 50 grams of support material may cost only \$1, but removing them can take an hour of labor work, which can cost \$20. Some processes, like DLP, may not even add time to print supports, making the post-processing the major concern in the increased cost. There are a few methods addressing the support removal problem, for example, weakening the supports to facilitate the removal [10] and using robotics to remove them [11]. However, a significant amount of time is still needed if there are many contact points, where the supports fuse to the model. It becomes extremely difficult when the contact points are located in some concavities of the model, like the one in Fig. 1b.

This motivates the present research to explore the possibilities to reduce the number of contact points and try not to have some supports fully surrounded by concave features. It is observed that the top of the support is unavoidably in contact with the model, but its base is attached to either the build plate or the model, depending on what is beneath. If the bases could be built not on the model, but only on the build plate (like Fig. 1c), there would be fewer contact points and all the structures, including those supporting concave features would at least have part of them exposed to the outside, facilitating their removal. However, the generation of tree-supports is already at least NP-hard [12]. Finding a tree that does not root in the model is even more challenging. This paper proposes a practical method to find such a solution – named Escaping Tree-Support (ET-Sup). Being practical here means that the method can determine if there is a solution and find it in a reasonable time. The main idea is to first classify all the points need to be supported as clear, obstructed, or enclosed. A clear point means that there is nothing below it except the build plate, an obstructed point is not clear, but it can reach the build plate through branches, and an enclosed point cannot reach the build plate at all. After that, the supports are constructed to link the obstructed points to the clear points, such that the obstructed points 'escape' from the model and come clear. The contributions are summarized as follows:

- Starting from the build plate (bottom-up), morphological operations that can consider the support size is applied across the layers to reach and classify the support points.
- A divide-and-conquer algorithm is employed to group the support points according to their height level (top-down) and connect them through branches based on their classification.
- A ray representation is used to make the above operations can be efficiently performed, including the clearance check and interference search.

Experimental results show that the present method can successfully generate ET-Sup for various models including the mushroom shown in Fig. 1. ET-Sup facilitates its removal and helps the 3D printing industries and communities reduce the manufacturing lead time. This paper focuses on the tree-support structures and the purpose is to ensure the printability. It is expected further research to develop more sophisticated algorithms for other type of supports and purposes.

The rest of the paper is organized as follows. The literature is reviewed in Section 2. Section 3 presents the technical details of the ET-Sup generation. The results are given in Section 4, and the paper concludes in Section 5 with discussion and future works.

2. Literature review

Since support structures are often unavoidable in additive manufacturing, there is growing attention on them. Interested readers can find the past development in a review paper [13]. This section surveys the more recent works in the related topics, in terms of the minimization, the detection, and the generation of support structures.

2.1. Support minimization

When it is still in the design phase, the shape of a model can be optimized to minimize the use of supports. For example, support slimming [2] deforms the overhangs of a model such that they become self-supported while preserving the global shape of the model. Reducing support structures can be formulated as a constraint in topology optimization [14]. Similarly, there is also flexibility to design support-free hollowing [15, 16] during the generation of internal structures. When the model cannot be altered any more and needs to be printed as is, advanced manufacturing techniques could be applied to print the part without supports. For example, Dai *et al.* [17] used a robotic arm to deposit material along curved layers, and Yang *et al.* [18] used a rotating optical fiber to cure materials at different angles. However, these would require specific tools and make their use limited to the experts. When multiple parts are fabricated in one build, the support materials can be reduced by putting a part under another part [19], but this creates even more contact points on the parts, increasing the risk of damages and the difficulty of removal.

2.2. Support detection

A straightforward way to detect the support points (i.e., overhangs) is to compare the facets' normal of a standard triangle language (STL) model with the build direction [20]. However, it is more complicated than imagine because there are also the self-support capability and the bridging effect of material, and this simple checking method may result in much more contact points than needed. As such, various methods were proposed to work with other representations. Huang *et al.* [21] computed the support regions in image space by subtracting the slice image of a layer by the one of the layer below, and Shi *et al.* [22] sliced on boundary representation (B-rep) models to get the support points. Huang *et al.* [23] used surface elements (surfel) to describe the points on a model's surface and applied machine learning for support detection, which can find much fewer support points. Recently, Jang *et al.* [24] applied layered depth images to detect support points, which can guarantee that no floating island would be missed. However, none of these methods can classify if a support point is obstructed or enclosed, which is an objective of this paper.

2.3. Support generation

To properly hold the detected support points, there are different support structures developed in the literature, including vertical array [8], lattice [25], slopping wall [26], bridge-like [27], and tree-like [28]. Among them, tree-like support is much considered since it groups the support branches before reaching the build plate and thus better utilizes materials. Generating tree-supports can be described as the Euclidean Steiner Minimal Tree problem, which is at least NP-hard [12]. Zhu *et al.* [9, 29] used a hybrid of a particle swarm optimization and a greedy algorithm to design the tree-supports. The method saves up to 8% material and 13% print time compared to the results obtained by the Autodesk Meshmixer [7], and it has a computation time from minutes to hours. They further stabilized the fabrication process and reduced the material waste by using a two-level support structure, where the first level uses beam-supports and the second level uses tree-supports [30]. To speed up the tree generation process, Zhang *et al.* [28] presented a divide-and-conquer (D&C) strategy to iteratively create branches through the local barycenters. This method reduces the computation time down to seconds and saves even more material up to 50%, compared to Meshmixer [7]. All these methods aim to reduce the material waste and the print time, but none made the supports escape from the model to reduce the post-processing time. Although Meshmixer [7] allows users to choose whether they want all supports to be built on the build plate only, it simply leaves the overhangs unsupported if it cannot find a solution for them. Properly achieving this function is another objective of this paper, even when the support points are obstructed.

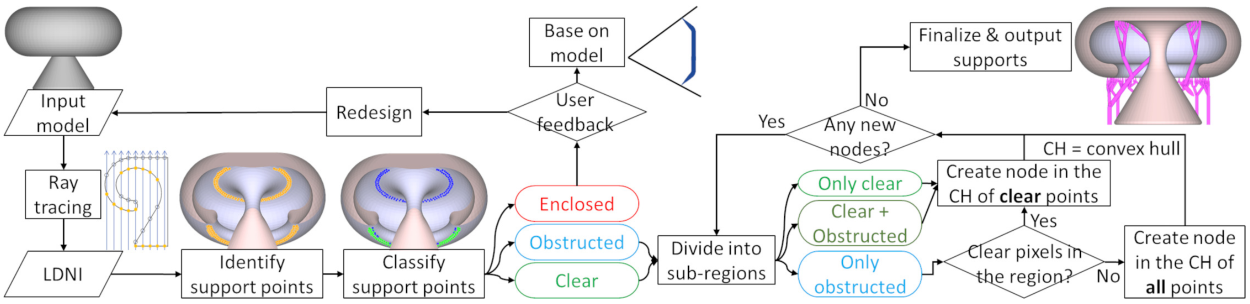

Figure 2: A flowchart of the ET-Sup generation process.

3. Methodology

The overview of the proposed ET-Sup generation is shown by a flowchart in Fig. 2. As mentioned in the introduction, the keys to realize the ET-Sup are the classification of support points and linking the obstructed points to the clear points. Both operations require the spatial relations of the support points. Therefore, a richer representation – layered depth-normal image (LDNI) – is employed. LDNI [31] is a ray representation containing the information as of voxel ones, but taking much less space, and it has been shown to be a robust tool for geometric operations [32]. It will be first demonstrated by the identification of support points and then applied to obtain the needed spatial relations for the other operations on-the-fly. The required inputs of the method besides the model are:

- layer thickness (t),
- self-support length (s) – the maximum horizontal length can be printed in one layer that is not supported by the previous layer,
- support clearance (c) – the support size, the gap between the support and the model, etc.

The self-support length is defined by the 3D printer and the material used, or by the angle threshold for overhangs (θ), i.e., $s = t \tan\theta$. The clearance controls how far the supports should be kept from the model, which is used to reduce the risk of the supports merging with the model and damaging the model. There are also other parameters like z-offset and tip size, but they are optional and have a predefined value.

3.1. Identification of support points

A LDNI is a two-dimensional (2D) image located on the build plate and large enough to cover the entire model (see Fig. 3). Through rendering techniques, each pixel (px) shoots a ray from the build plate to intersect with the model and stores a sequence of sample points (p) with a depth and a normal, i.e., $p = (i, j, d, \mathbf{n})$, where

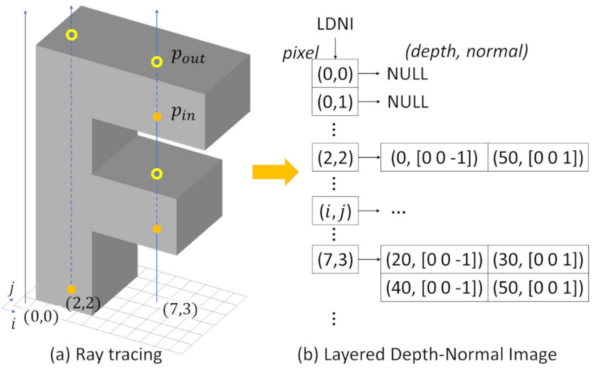

Figure 3: An illustration of converting a letter ‘F’ model to a LDNI.

i and j are the pixel indices, d is the depth, and $\mathbf{n}$ is the normal vector. Thus, the LDNI sample points can give information along the height, without going through all the layers or doing 3D intersections. For example, they can be used to quickly detect whether a position (x, y, z) is inside or outside (In/Out) the model. Assume the build plate is located in the $x - y$ plane and the build direction is along the z-axis, the In/Out check can be done by first getting the pixel (i, j) where (x, y) is located and then comparing the z value to the depth (d) of the sample points on the ray of the pixel. If it is found above a sample point where the ray enters the model (in-point, p_{in}) and below a sample point where the ray exits the model (out-point, p_{out}), then the position (x, y, z) is located inside the model. The In/Out of a sample point can be classified easily by its normal, i.e., if its normal points towards the build plate, it is an in-point.

Given the LDNI (I), Algorithm 1 shows how to locate the points need to be supported. Firstly, only the in-points (p_{in}) need to be checked, which can be collected by going through all the rays and checking the normal of all sample points. Secondly, one way is to use the normal and compare with the build direction to check if a point needs to be supported, but this method overlooks the self-support capability of material and identi-

Algorithm 1: Identify support points

```
   input  : LDNI ($I$), self-support length ($s$)
   output: $SupPointList$
1  forall in-point $p_{in}$ : $(i, j, d)$ of $I$ do
2      $l \leftarrow layer(d)$;
3      $bSup \leftarrow false$;
4      forall $px : (m, n) \in I \mid dist(px, p_{in}) \leq s$ do
           /* $dist()$ returns 2D distances */
5          if $I.io(m, n, l)$ and $I.io(m, n, l-1)$ then
               /* $io()$ checks In/Out */
6              $bSup \leftarrow true$; break;
7          end
8      end
9      if $bSup = false$ then
10         $SupPointList.push(p_{in})$
11     end
12 end
```

fies more support points than necessary. Given the self-support length (s), Algorithm 1 checks if there is any material in the previous layer ($l - 1$) within the self-support length centered at the point p_{in}. The function $io()$ returns true if a position is inside the model. The layers should be connected to provide the support, so there should be material at the same position too in the current layer (l) where p_{in} is located. Finally, if no self-supporting is found, the point is stored in a list of support points ($SupPointList$), which is used for the next steps.

The number of rays and sample points depends on the image resolution, i.e., the higher resolution, the more rays and points. There is a balance between the sampling error and the computation speed. In the context of support generation, the resolution only needs to be as high as the required density of the support points. In other words, each support point can support a portion of the overhangs, and the size of the portion is related to the self-support length (s), so the distance between two support points can be as large as $2s$. Similarly, the pixel size of the image can be $2s$ as well. To be safe and to get a smoother result, this paper uses a smaller pixel size, i.e., $0.5s$, and sub-samples the support points to maintain the distance between points roughly equal to $2s$.

Implementation note

The search of the self-support region in line 4 of Algorithm 1 is a simplified version just to make the algorithm easier to understand. This way does not consider the connectivity between the pixels, and there may be a risk that a supported pixel (m, n) is not linked to the point (i, j). In the actual implementation, a front propagation method is used to make sure there is a connection to every pixel that is checked for self-supporting.

3.2. Classification of support points

A key component to the success of building ET-Sup is knowing which support points are reachable by the support structures that have their bases on the build plate. This paper classifies the support points into three categories:

1. **Clear**: can be directly extruded to the build plate
2. **Obstructed**: need to be displaced until clear or be linked to a clear point
3. **Enclosed**: can only add support onto the model

Examining the clearness of a point is straightforward, i.e., iterating over all pixels inside a circle of radius c centered at the point and checking if there is anything below them. The challenge is how to identify the obstructed points from the enclosed ones, because both are not clear and it requires the spatial relation between the points and the whole model to distinguish them. A propagation algorithm and morphological operators are developed in this paper to make this classification fast. Note that the support points are commonly offset by a small distance in the normal direction of the surfaces they are located, and a cone is built in the offset distance, so that the supports are always approaching to the surface in its normal direction and the support removal is easier. Therefore, it is actually the offset support points here is classifying.

The method to classify the obstructed and the enclosed points is illustrated in Fig. 4 and detailed in Algorithm 2. The basic idea is to grow a feasible region (FR) from the build plate all the way to the top and see which support points are reached. This is realized by two morphological operations and with the help of a supported region (SR), which is the region supported by the feasible region of the previous layer. For the first layer (SR_0), the supported region is set as the whole build plate. The first morphological operation is a dilation that enlarges the feasible region (FR_l) by the self-support length s to get the supported region (SR_{l+1}). The second operation is like an erosion that shrinks the supported region (SR_l) to obtain the feasible region (FR_l). However, the shrinkage needs to be based on how the model occupies the space, so this erosion-like operation is implemented by a subtraction and two dilation operations. Firstly, since only the supported region and the region around are of interest, this region

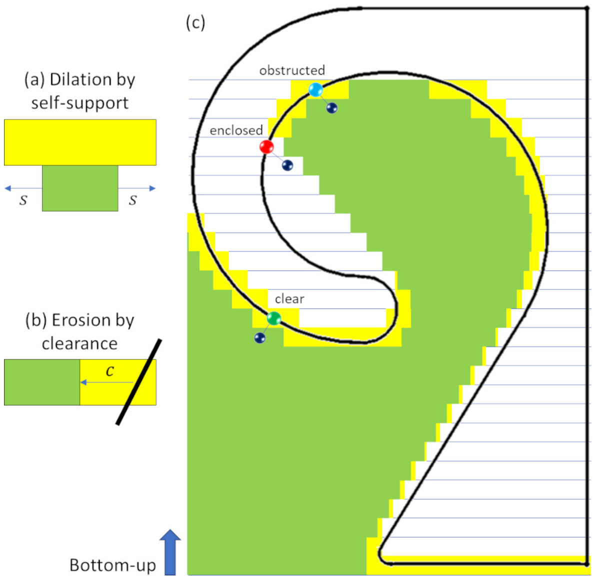

Figure 4: Support classification: (a) A dilation operation is applied to the feasible region (green) to get the supported region (yellow) for the upper layer. (b) An erosion operation is applied to the supported region to get the feasible region for the same layer. (c) An illustration of the process applying the dilation and the erosion operations repeatedly starting from the build plate to reach the support points.

```
Algorithm 2: Classify support points
input  : $SupPointList, I, s, c$
output: $ObstructList, EncloseList$
1  $L \leftarrow \max_{layer}(SupPointList)$;
2  $SR_0 \leftarrow$ build plate ;      // Supported region
3  for $l = 0$ to $L$ do
       /* Erosion-like operation          */
4      $IR \leftarrow \emptyset$ ;                    // In region
5      forall $px : (m, n) \in (SR_l \oplus c)$ do
6          if $I.io(m, n, l)$ then
7              $IR.push(px)$;
8          end
9      end
10     $FR_l \leftarrow SR_l - (IR \oplus c)$ ; // Feasible reg.
       /* Classification                  */
11     forall $p \in SupPointList \mid layer(p) = l$ do
12         if $p \in FR_l$ then
13             $ObstructList.push(p)$
14         else
15             $EncloseList.push(p)$
16         end
17     end
       /* Dilation operation              */
18     $SR_{l+1} \leftarrow FR_l \oplus s$
19 end
```

of interest is acquired through applying a dilation operation to the supported region by the clearance c. Secondly, the in-region (IR) that the model occupies can be found by checking the In/Out for each pixel in the dilated region. After that, another dilation is applied to the in-region IR to consider the clearance c. Finally, the result is subtracted from the supported region SR to get the feasible region FR at the same layer. These two operations are iterated layer-by-layer until the layer of the highest support point is checked. All the non-clear support points that are reached by the feasible region are the obstructed points, otherwise they are the enclosed points. This method can quickly classify all the points with one single pass of propagation, and it can also consider the support clearance (c) in the search.

Implementation note

The layers are relatively quite small compared to the model, and thus the shape change in the consecutive layers could be insignificant. Therefore, the search does not have to be done in every layer, but only every n-layers (e.g., $n = 50$), as long as the self-support length (s) and the support clearance (c) are multiplied by n too. The exceptions are the layers which contain support points and where the model has sharp change in shape. These layers must be checked in order not to miss any support points or grow the feasible region wrongly. The sharp change in shape can be detected quickly using LDNI too through the analysis of a shape profile [33]. In addition, because the In/Out check is always from the bottom to the top, a 2D array can be used to record the checked sample point on each ray of the LDNI. In this way, each In/Out check is further sped up, from $O(k)$ to $O(1)$, where k is the number of sample points on a ray.

3.3. Generation of escaping tree supports

With the support points classified, they can be linked with branches according to their combinations. To have an efficient approach, the D&C method [28] is considered in this paper to generate the tree structure. Although this method can group the support points quickly, it is not able to build structures that escape from the model, so it cannot be directly applied here. In addition, the previous method processed the support points by the overhang regions sequentially, which can make sure the points are close to each other and have similar property (e.g., height). However, this limits the grouping to happen only in each region and eliminates many combinations that could lead to better solutions. Therefore, the same D&C strategy is employed here to divide the support points into different sub-domains and build

Algorithm 3: Scan points by height

```
input : $SupPointList$, $I$
output: $Support$
1  $L \leftarrow \max_{layer}(SupPointList)$;
2  $mask \leftarrow \emptyset$ ;                          // 2D Array
3  for $l = L$ to 0 do
4      forall $p : (i, j) \in SupPointList \mid layer(p) = l$ do
5          if $mask(i, j) \neq \emptyset$ then
6              $Support$ += $GenerateSupport(mask, l + 1)$;
7          end
8          $mask(i, j) \leftarrow p$;
9      end
10 end
11 $Support$ += $GenerateSupport(mask, 0)$;
12 $Support$ += $BuildSupportToPlate(mask)$;
```

the tree structures partially in each sub-domain, but a completely new dividing and construction method is developed for the generation of the ET-Sup.

The support points here are in the form of the LDNI sample points. Although they do not have the information about whether they are from the same overhang region, their spatial relationship can be obtained through the image space. That is, the division can be done by splitting the image into equally sized blocks. In this way, the support points can be grouped regardless of their connectivity on the model. However, since all the support points are considered together, there could be overlapping (i.e., more than a point located in a pixel), and some points may differ very much in height even they belong to the same block. Therefore, a top-down scanning algorithm is developed to make sure the support points are processed in order, which is detailed in Algorithm 3. This scanning algorithm collects support points to a 2D mask (*mask*) layer-by-layer from the highest (L). The mask has the same size of the LDNI, and the points (p) are stored in the same pixel location (i, j). This mask is maintained always having only one point in each pixel. When another point is going into an occupied pixel, it will pause the scanning at the height and generate the support structures partially for the points above ($GenerateSupport()$). This partial structure is restricted not to go lower than the height $(l + 1)$ where the scanning is paused.

The overall D&C approach to generate the support structure is shown in Algorithm 4. In the dividing step ($DivideSubRegions()$), the mask is divided into a set of blocks with a size h_g. The size of blocks is small at the

Algorithm 4: Generate Support

```
input : $mask$, $l_{\min}$, $s$, $ClearList$, $ObstructList$
output: $Support$ (partial)
1  $h_g \leftarrow 2s$;
2  while $h_g < size(I)$ do
3      $SubR \leftarrow DivideSubRegions(mask, h_g)$;
4      forall $r \in SubR$ do
5          $RoI \leftarrow \emptyset$ ;   // region of interest
6          switch $\chi : \{\forall p \in r\}$ do
7              case $\chi \subset ClearList$ do
8                  $RoI \leftarrow CH(\chi)$ ;  // conv hull
9              case $\chi \subset ObstructList$ do
10                 $\zeta \leftarrow FindClearRegion(r)$;
11                 if $\zeta \neq \emptyset$ then
12                     $RoI \leftarrow \zeta$;
13                 else
14                     $RoI \leftarrow CH(\chi)$;
15                 end
16             otherwise do // clear+obstruct
17                 $RoI \leftarrow CH(\chi \cap ClearList)$;
18             end
19         end
20         $p_{new} \leftarrow FindValidNode(RoI, l_{\min})$;
21         if $p_{new} \neq \emptyset$ then
22             $Support$ += $Connect(p_{new}, \forall p \in r)$;
23         end
24     end
25     $h_g \leftarrow 2h_g$;
26 end
```

start, i.e., two times the self-support length (s), and it is doubled in each iteration. Each block is a sub-region ($SubR$), and the conquering step is performed to connect the support points in each sub-region (some examples can be seen in Fig. 5). The main goal here is to find a pixel, preferably clear, within each block at a height such that it is reachable by all support points through self-supported branches and will not cause any collision to the model. The collision detection can be done quickly by the In/Out check in the image space. If there are multiple solutions, the one that uses the least material is selected. This search ($FindValidNode()$) can be realized by checking all the pixels within the block and lowering the height by a layer repeatedly until such a pixel is found or the height goes below the limit. However, this way would have many unnecessary checks when the blocks are large. Therefore, a region of interest (RoI) is introduced to speed up the process, and the search will be performed only in the RoI. The RoI is set up based on the classifications of the support points

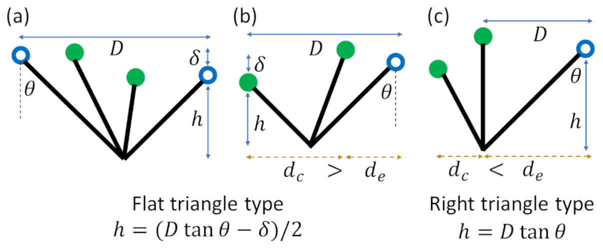

Figure 5: Two types of connection to create branches: flat triangle vs. right triangle, and their way to calculate the height (h). The solid points (green) represents the clear points, and the hollow points (blue) represents obstructed points.

(χ). In the case of all clear points ($\chi \in ClearList$), the RoI is the convex hull of these clear points ($CH(\chi)$). In the case of all obstructed points ($\chi \in ObstructList$), it first tries to find whether there are any clear regions within the block ($FindClearRegion()$). If yes, the RoI is set to these clear regions (ζ), otherwise the RoI is the convex hull of the points and the new node maintains an obstructed status. In the case of having both clear and obstructed points, the RoI is the convex hull of only the clear points. Once a valid pixel is found, a new node (p_{new}) is created at the pixel's location, and all the support points in the block are connected to it forming branches ($Connect()$).

After generating the partial tree-supports, there is a function $BuildSupportToPlate()$ at the end of Algorithm 3 to connects all the latest new nodes (p_{new}) to the build plate. These are the nodes can no longer be grouped to form branches due to their heights being too low or connecting them will make collision to the model. If a nodes is clear, a vertical pillar can be built to connect it directly to the build plate. If a node is obstructed, it finds a shortest path to reach a clear location using the propagation algorithm applied to support classification in Section 3.2. This would be computationally expensive, but thanks to the nature of the tree-supports, only a limited number of nodes near the bases need to be processed, and surprisingly none of the tested cases (even the mushroom model) need this operation. Once the whole tree connectivity is defined, the final support structures can be obtained by putting a cylinder with a user-defined size to each of the branches.

Implementation note

For the function $FindValidNode()$ to find a valid node, it does not have to start from the layer of the lowest point within the region and lower the height one layer at a time. Based on the configuration of the points, it is possible to estimate the height where a valid point is more likely to be found. There are two different types of connection to create branches as shown in Fig. 5. Generally, when the points are evenly distributed, the farthest points are connected together through the lines inclined at the self-support angle (θ). In this case, the shape of a flat triangle is formed, and the height (h) below the lower point can be calculated as: $h = (D \tan\theta - \delta)/2$, where D and δ are the horizontal and vertical distances between the farthest points. When an obstructed point is far away from the clear points, it should be connected to the nearest point at the inclined angle, which results in the shape of a right triangle, and the height is calculated as: $h = D \tan\theta$, where D in this case is the horizontal distance between the obstructed point and its nearest clear point. In this way, most of the valid points were found at the first height being checked or among a few layers under that height.

4. Results

The present methodology is implemented in C++ on top of the online available source code of LDNI [31]. It is tested on a PC running 64-bit Windows 10 equipped with Intel Core i5-6500 CPU@3.20GHz, 8GB RAM, and NVIDIA Quadro K620. Three examples are used to demonstrate the method, and the time statistics are also reported. The results are mainly compared with the commercial software – Autodesk Meshmixer [7]. Most settings are using the default values for Ultimaker, but since Meshmixer cannot consider the self-support length, 'Angle Thresh' (i.e., self-support angle) and 'Density' are reduced from 45° and 75% to 30° and 50%, respectively, to mimic the self-support capability. This change only reduces the number of support points and thus improves the computational speed for Meshmixer, so it does not exist any unfairness in the comparison. 'Post Diameter' is also reduced from 3 mm to 1 mm for the sake of saving material. There is an option of 'Allow Top Connections' to switch between whether the support bases can be located on the model or must be on the build plate. This feature will be tested as well, and it will be shown that it is very limited in practice. For the required inputs of the present method, the layer thickness (t), the self-support length (s), and the support clearance (c) are set as follows: $s = 1$ mm, $c = 1$ mm, and $t = 0.1$ mm.

4.1. Letter 'F'

In this section, a letter 'F' model is used to demonstrate the present method. A step-by-step illustration in Fig. 6(a) shows how the ET-Sup is generated after the support points are identified, offset, and clas-

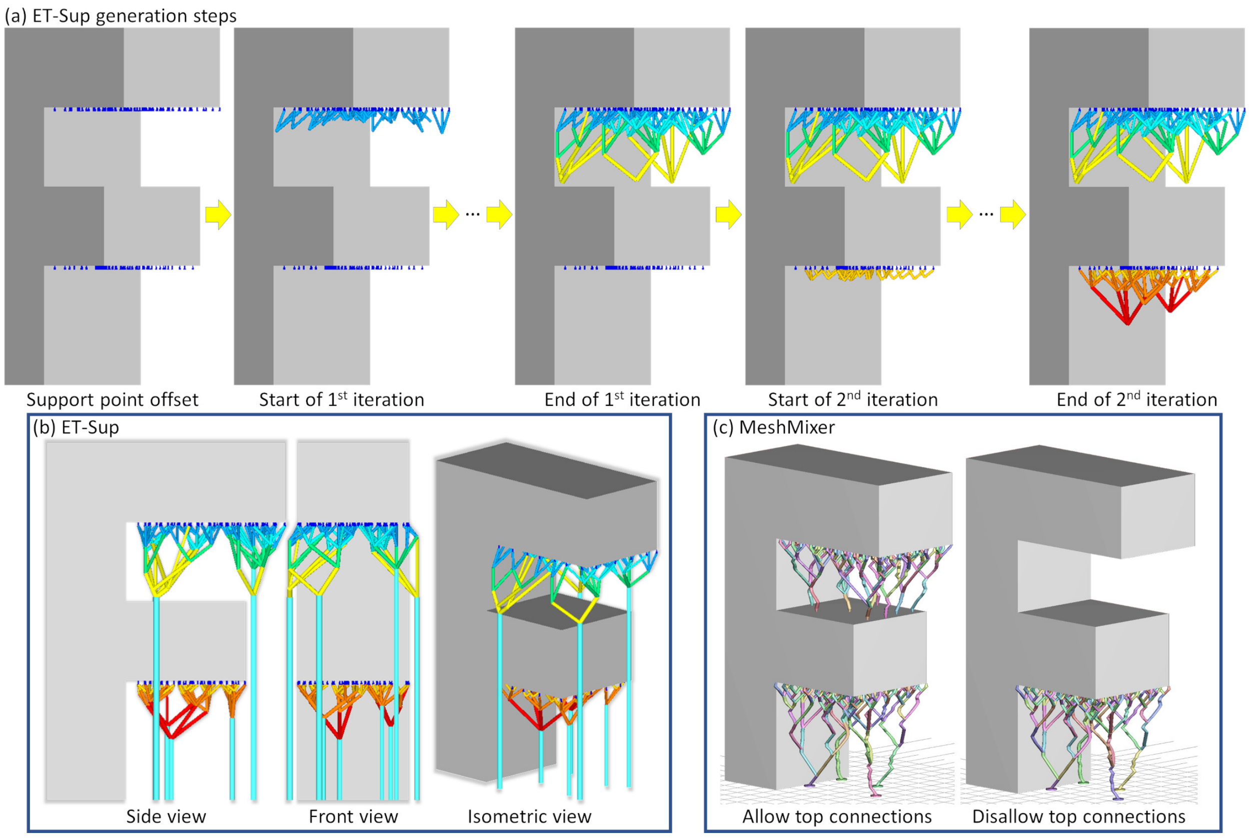

Figure 6: (a) The steps of generating ET-Sup. (b) The final result of ET-Sup in different viewing angles. (c) The tree-support generated by Meshmixer with and without top connections (i.e., root in model).

sified. The letter 'F' model has two horizontal overhangs: one at the top and one in the middle, and the support points are located at the bottom of them. The support points at the top are classified as obstructed, except the ones at the tip, and all the other points are clear. One iteration in the figure is one call of the function $GenerateSupport()$ in Algorithm 3 to partially generate support structures when there are overlapping support points in different height levels. As such, the first iteration builds support for the top overhang, and the second iteration builds support for the below one. The function $GenerateSupport()$ is detailed in Algorithm 4, where the divide-and-conquer (D&C) takes place to create supports in each sub-region. The supports that are generated in different D&C steps are highlighted in distinct colors. Usually, the new nodes from higher height levels would be linked to the points in the lower ones, but it is not seen in this example because linking them would cause intersections to the model. At the end of the iterations, the last nodes are linked to the build plate, and the final result is shown in Fig. 6(b).

From the front view, it is clear that the obstructed points at the top have their supports go towards either the left or the right side of the model where they become clear. This phenomenon is caused by two cases of handling obstructed points in the D&G algorithm. First, for the sub-regions on the edge, when all the points are obstructed, the function $FindClearRegion()$ returns the nearby region that is out of the model region. A new node is created in this clear region and thus the node is clear. For other sub-regions containing all obstructed points, a new node is created roughly in the center of the points and remain obstructed. Second, when the sub-regions get bigger in the subsequent D&G steps, the obstructed nodes are grouped with the clear nodes. It becomes the case of 'Clear + Obstructed', and a new node is created based on the location of the clear nodes. This case is applied similarly to the other obstructed nodes in the middle when the sub-regions are larger and larger. As a result, the supports are constructed from the model region to somewhere outside, which looks like they are escaping from the model.

The tree-supports generated by Meshmixer are shown in Fig. 6(c) with both allowing and disallowing top con-

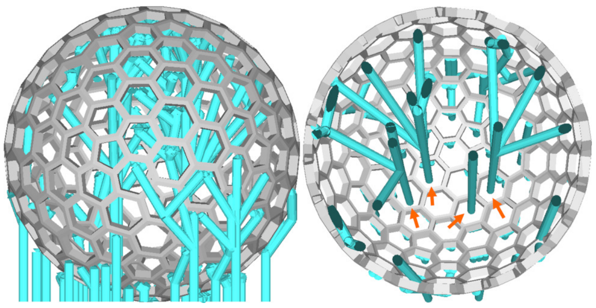
Figure 7: Lattice ball with ET-Sup.

nections. On the one hand, when top connections are allowed, the supports at the top have both ends touching the model, which would unnecessarily damage the top surface of the lower overhang. On the other hand, when top connections are disallowed, Meshmixer tries to join the supports such that they do not touch the model, otherwise they are not constructed. In this case, all the supports at the top are eliminated, leaving the overhang unsupported, which is obviously not a valid solution. Therefore, although Meshmixer has the option not to build the support bases on the model, its algorithm cannot search for the escaping paths to realize the ET-Sup.

4.2. *Lattice ball*

A lattice ball shown in Fig. 7 is used to test the capability of the present method in handling complex topology. The lattice ball is made of honeycomb structures over the surface of a ball, and it has a genus number of 190, e.g., the number of holes. It can be expected to be very tedious if one needs to remove the supports from each of the holes. Similar to a hollow ball shape, the inner surface of the top half and the outer surface of the bottom half are the surfaces that face downward and need to be supported. The ET-Sup has been successfully generated for all the support points, and the support structures are built on the build plate through the holes of the lattice without intersecting with the model. This example verifies the generality of the approach, and it can work for models with various complexity.

4.3. *Mushroom*

The mushroom model is already shown in Fig. 1. Its cap has a deep concavity, which could enclose much support material in it and make the support removal challenging. This example compares with the results of two other support generation methods. One is the linear support generated by Cura [6], and the other one is the tree-support generated by Meshmixer [7]. Apparently, the ceiling of the cap cavity needs to be supported, and

Table 1: Material use and print time of different supports for the mushroom model, where g = gram, h = hour, and m = minute.

	Material	**Print time**
Mushroom	20 g	2 h 40 m
Linear support	6 g	1 h 2 m
Tree-support	2 g	2 h 37 m
ET-Sup	1 g	1 h 2 m

all methods generate support inside the cap. Both the linear and the tree supports have their tops supporting the ceiling and the bases touching the floor of the cap cavity. Surprisingly, even in such deep concavity, the present ET-Sup can find its way out from the cap cavity and join the supports below the cap. As a result, while it was difficult to remove the linear and the tree supports without breaking the stem of the mushroom, the ET-Sup were successfully removed by a usual forceps.

In addition, with the consideration of self-supporting and the ease of support removal, the supported surfaces of ET-Sup have a better finishing even just assessed by eye (see Fig. 1). To quantitatively measure the surface roughness, a Phase II+ SRG-4000 Surface Roughness Tester is used to obtain the profile of supported surfaces with a sampling length of 0.25 mm. The arithmetic mean roughness (*Ra*) is reported, which indicates the average of the absolute value along the sampling length. The linear support and the tree-support have a *Ra* value of 31.93 μm and 29.79 μm, respectively, while the ET-Sup has a *Ra* value of 19.65 μm. This results are in line with the human visual assessment.

In terms of the material consumption and print time, the statistics are summarized in Table 1. The mushroom model itself uses 20 grams (g) of material and takes 2 hours (h) and 40 minutes (m) to print by a FFF printer – Ultimaker. For each of the supports, the table shows the extra material and the print time added to the print. For example, the linear support takes an extra of 6 g material and 1 h 2 m print time, i.e., a total of 26 g material and 3 h 42 m print time. Oddly enough, the tree-support uses much less material (2 g) but has a much longer print time (2 h 37 m). The reason behind is that although the linear support needs much more material, they are printed continuously; in contrast, the print head needs to travel all around the model to print the tree-support, and thus there are many empty travels resulting in a longer print time. The ET-Sup uses even less support material (1 g) than the commercial software. This means that the previous work does not generate an optimal tree, and the ET-Sup is an improvement even escaping supports suppose to use more material.

Table 2: Time statistics of generating ET-Sup and tree-support (by Meshmixer) on the tested models. m = min, s = second.

Model	size (mm)	Identify point	Classify point	Generate support	**Total**	**Meshmixer**
Letter ‘F’	$60 \times 28 \times 90$	0.071 s	0.050 s	0.390 s	0.511 s	30 s
Lattice ball	$60 \times 60 \times 60$	0.061 s	0.375 s	0.168 s	0.604 s	4 m 15 s
Mushroom	$53 \times 53 \times 37$	0.052 s	0.016 s	1.527 s	1.595 s	1 m
Teeth	$42 \times 70 \times 81$	0.016 s	0.433 s	0.504 s	0.953 s	3 m 45 s

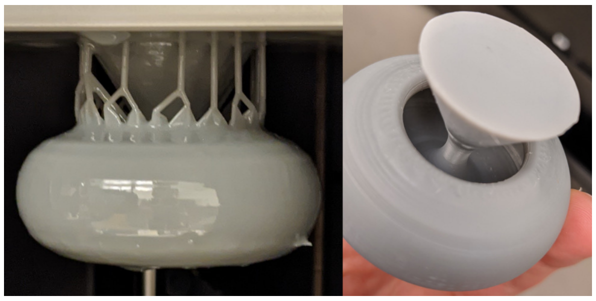
Figure 8: The mushroom model is fabricated by a bottom-up DLP printer (3D Systems FabPro 1000).

One can use multi-material FFF to print the supports with soluble materials. However, other manufacturing processes like digital light processing (DLP) can hardly print different materials and thus the supports must use the same material. To test the present method in different manufacturing processes, it is also applied to generate ET-Sup for DLP to print the mushroom model. The DLP printer used is the 3D Systems FabPro 1000, which is a bottom-up process, i.e., the object is printed upside-down. The result is shown in Fig. 8, and the supports were successfully removed from the cap cavity as well with a usual forceps. This verifies the applicability of the ET-Sup technique.

4.4. Dental model

To demonstrate the social impact of the method, a maxillary teeth model is tested. One one hand, this kind of model normally has some critical surfaces (e.g., teeth), and the model should be oriented such that those surfaces have the least supports. On the other hand, certain 3D printing processes (e.g., SLA/DLP) require printing a model at an angle to increase the success rate and improve the surface finishing. To satisfy both requirements, it would be inevitable to have some parts of the model on top of the other parts. This could create unnecessary contact points on the upward facing surfaces, and preventing them as well might result in no solution. The present method can eliminate these unnecessary points and thus can alleviate the challenges in finding a suitable print orientation. In this example, the model is printed on its side vertically with a 10° tilt

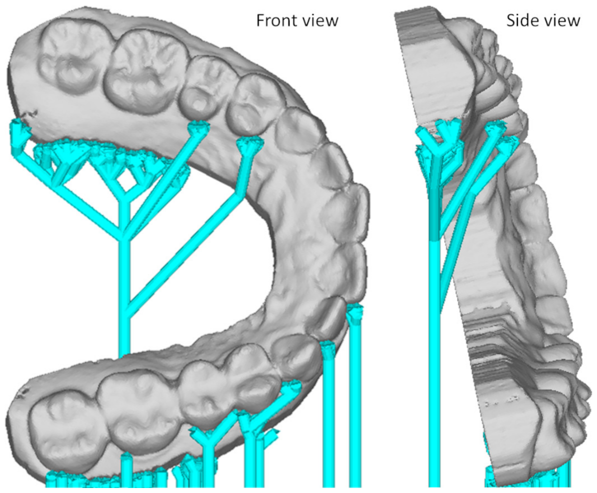

Figure 9: Teeth model with ET-Sup.

as shown in Fig. 9. Most support points are located at the sides of the base, and a few at the sides of the teeth. Many of the upper support points are obstructed by the lower part of the model, and if vertical pillars are used to support them, they will be touching the critical surfaces of the lower part. The ET-Sup has been successfully generated for all the support points, and the support structures are only built on the build plate. This example verifies the practicality of the approach, and it has a great potential for many other applications.

4.5. Time statistics

The statistics of computation time for the three reported models are listed in Table 2. Besides the total time, the time of each sub-process is also reported, namely the identification and classification of support points as well as the generation of ET-Sup. The time of point identification also includes the time for ray tracing to obtain the LDNI representation. Due to the reason that no other methods can generate ET-Sup, there is no comparison with other algorithms that work for the same goal. However, the table also includes the computation time of Meshmixer [7], which only generates tree-supports. It can be observed that the time varies a lot in different models for Meshmixer. It takes

30 seconds for the letter ‘F’ model, which has a simple shape, but it takes more than 4 minutes on the lattice ball model. This reveals that the algorithm highly depends on the shape complexity. Thanks to the LDNI, the shapes are represented by sample points on the rays, and they can be processed in a unified way regardless of the shape complexity. This can be seen from the results that all the examples are completed in a similar time (0.5 – 1.5 seconds). In addition, the divide-and-conquer method [28] also took around 1 second to generate tree-supports in all their examples. This shows that the present method is comparable with the state-of-the-art tree-support generation methods, even though the search for escaping paths should take a longer time.

5. Conclusion

This paper presents a new support structure that tries to build the support bases on the build plate to minimize the number of contact points. The support finds its way out of the model even the support points are located in a deep concavity, and thus it is named escaping tree-support (ET-Sup). To realize the ET-Sup, this paper proposes to classify the support points into three categories: clear, obstructed, and enclosed. There is no solution for the enclosed points, but the obstructed points can be, for example, grouped with the clear points to escape from the model. The classification is realized by a propagation algorithm, and the grouping is implemented in a divide-and-conquer method. The efficiency of these operations is achieved with the help of a ray representation – layered-depth normal images (LDNI). The experimental results have verified the present method, and its computational speed is as fast as the state-of-the-art and much faster than the current commercial software, although they only generate tree-supports.

However, ET-Sup is not always the best solution. Since ET-Sup is developed to facilitate material removal in deep concavities, it does not have much benefit in simple models. If a model does not have complex overhangs or concave features, ET-Sup is essentially a tree-support. In addition, the linear support is better for simple models in terms of stability and print time. There are also other limitations in the current implementation. For example, the final step of linking the supports to the build plate may result in some long and thin trunks that can cause stability issues. The support strength should be considered, and it could be enhanced by changing the support size or using networked supports. In regard to the contact points between the supports and the model, a tip is currently placed on each of them to facilitate detaching the supports from the model. However, a tip still has a strong connection, which requires certain tools like a forceps to detach it. With the ET-Sup, it is preferred to design some disassembly features such that the contact points can be detached automatically or easily by a simple twist or push, and then the whole supports can be drawn out from the base. User preferences can be incorporated too, if in case there is some particular holes (like in the example of lattice ball) that the users want the supports to come out from.

References

[1] X. Zhang, X. Le, A. Panotopoulou, E. Whiting, C. C. L. Wang, Perceptual models of preference in 3D printing direction, ACM Trans. Graph. 34 (6) (Oct. 2015). doi:10.1145/2816795.2818121.

[2] K. Hu, S. Jin, C. C. L. Wang, Support slimming for single material based additive manufacturing, Computer-Aided Design 65 (2015) 1 – 10. doi:10.1016/j.cad.2015.03.001.

[3] J. Jin, Y. Chen, Highly removable water support for stereolithography, Journal of Manufacturing Processes 28 (2017) 541 – 549. doi:10.1016/j.jmapro.2017.04.023.

[4] W. Ameen, M. Khan Mohammed, A. Al-Ahmari, Evaluation of support structure removability for additively manufactured ti6al4v overhangs via electron beam melting, Metals 9 (11) (2019). doi:10.3390/met9111211.

[5] L. Zhu, R. Feng, J. Xi, P. Li, X. Wei, A lightweight design of tree-shaped support structures for slm additive manufacturing, Computer-aided Design and Applications 17 (2019) 716–726. doi:10.14733/cadaps.2020.716-726.

[6] Cura, accessed: 2020-10-13 (2020). [link].
URL `https://ultimaker.com/cura`

[7] R. Schmidt, N. Umetani, Branching support structures for 3D printing, in: ACM SIGGRAPH 2014 Studio, SIGGRAPH ’14, New York, NY, USA, 2014, p. 1, in Autodesk Meshmixer. doi:10.1145/2619195.2656293.

[8] J. Lee, K. Lee, Block-based inner support structure generation algorithm for 3D printing using fused deposition modeling, Int J Adv Manuf Technol 89 (5) (2017) 2151 – 2163. doi:10.1007/s00170-016-9239-3.

[9] L. Zhu, R. Feng, X. Li, J. Xi, X. Wei, Design of lightweight tree-shaped internal support structures for 3D printed shell models, Rapid Prototyping Journal 25 (9) (2019) 1552 – 1564. doi:10.1108/RPJ-04-2019-0108.

[10] R. Vaidya, S. Anand, Optimum support structure generation for additive manufacturing using unit cell structures and support removal constraint, Procedia Manufacturing 5 (2016) 1043 – 1059. doi:10.1016/j.promfg.2016.08.072.

[11] S. Nelaturi, M. Behandish, A. M. Mirzendehdel, J. de Kleer, Automatic support removal for additive manufacturing post processing, Computer-Aided Design 115 (2019) 135 – 146. doi:10.1016/j.cad.2019.05.030.

[12] J. Vanek, J. A. G. Galicia, B. Benes, Clever support: Efficient support structure generation for digital fabrication, Computer Graphics Forum 33 (5) (2014) 117–125. doi:10.1111/cgf.12437.

[13] J. Jiang, X. Xu, J. Stringer, Support structures for additive manufacturing: A review, Journal of Manufacturing and Materials Processing 2 (4) (2018). doi:10.3390/jmmp2040064.

[14] A. M. Mirzendehdel, K. Suresh, Support structure constrained topology optimization for additive manufacturing, Computer-Aided Design 81 (2016) 1 – 13. doi:10.1016/j.cad.2016.08.006.

[15] W. Wang, Y. Liu, J. Wu, S. Tian, C. C. L. Wang, L. Liu, X. Liu, Support-free hollowing, IEEE Transactions on Visualization and Computer Graphics 24 (10) (2018) 2787–2798. doi:10.1109/TVCG.2017.2764462.

[16] S. Choi, J. Ryu, M. Lee, J. Cha, H. Kim, C. Song, D.-S. Kim, Support-free hollowing with spheroids and efficient 3D printing utilizing circular printing motions based on voronoi diagrams, Additive Manufacturing 35 (2020) 101254. doi:10.1016/j.addma.2020.101254.

[17] C. Dai, C. C. L. Wang, C. Wu, S. Lefebvre, G. Fang, Y.-J. Liu, Support-free volume printing by multi-axis motion, ACM Trans. Graph. 37 (4) (Jul. 2018). doi:10.1145/3197517.3201342.

[18] Y. Yang, X. Li, X. Zheng, Z. Chen, Q. Zhou, Y. Chen, 3D-printed biomimetic super-hydrophobic structure for microdroplet manipulation and oil/water separation, Advanced Materials 30 (9) (2018) 1704912. doi:10.1002/adma.201704912.

[19] J. Jiang, X. Xu, J. Stringer, Optimisation of multi-part production in additive manufacturing for reducing support waste, Virtual and Physical Prototyping 14 (3) (2019) 219–228. doi:10.1080/17452759.2019.1585555.

[20] X. Huang, C. Ye, S. Wu, K. Guo, J. Mo, Sloping wall structure support generation for fused deposition modeling, Int J Adv Manuf Technol 42 (2009) 1074. doi:10.1007/s00170-008-1675-2.

[21] P. Huang, C. C. L. Wang, Y. Chen, Algorithms for Layered Manufacturing in Image Space, in: Advances in Computers and Information in Engineering Research, Volume 1, ASME Press, 2014, pp. 377–410. doi:10.1115/1.860328_ch15.

[22] K. Shi, C. Cai, Z. Wu, J. Yong, Slicing and support structure generation for 3D printing directly on b-rep models, Visual Computing for Industry, Biomedicine, and Art 2 (1) (2019) 2524 – 4442. doi:10.1186/s42492-019-0013-x.

[23] J. Huang, T.-H. Kwok, C. Zhou, W. Xu, Surfel convolutional neural network for support detection in additive manufacturing, Int J Adv Manuf Technol 105 (9) (2019) 3593 – 3604. doi:10.1007/s00170-019-03792-1.

[24] S. Jang, B. Moon, K. Lee, Free-floating support structure generation, Computer-Aided Design 128 (2020) 102908. doi:10.1016/j.cad.2020.102908.

[25] B. Vaissier, J.-P. Pernot, L. Chougrani, P. Véron, Genetic-algorithm based framework for lattice support structure optimization in additive manufacturing, Computer-Aided Design 110 (2019) 11 – 23. doi:10.1016/j.cad.2018.12.007.

[26] S. Cacace, E. Cristiani, L. Rocchi, A level set based method for fixing overhangs in 3D printing, Applied Mathematical Modelling 44 (2017) 446 – 455. doi:10.1016/j.apm.2017.02.004.

[27] J. Dumas, J. Hergel, S. Lefebvre, Bridging the gap: Automated steady scaffoldings for 3D printing, ACM Trans. Graph. 33 (4) (Jul. 2014). doi:10.1145/2601097.2601153.

[28] N. Zhang, L.-C. Zhang, Y. Chen, Y.-S. Shi, Local barycenter based efficient tree-support generation for 3D printing, Computer-Aided Design 115 (2019) 277 – 292. doi:10.1016/j.cad.2019.06.004.

[29] L. Zhu, R. Feng, X. Li, J. Xi, X. Wei, A Tree-Shaped Support Structure for Additive Manufacturing Generated by Using a Hybrid of Particle Swarm Optimization and Greedy Algorithm, Journal of Computing and Information Science in Engineering 19 (4), 041010 (06 2019). doi:10.1115/1.4043530.

[30] R. Feng, X. Li, L. Zhu, A. Thakur, X. Wei, An improved two-level support structure for extrusion-based additive manufacturing, Robotics and Computer-Integrated Manufacturing 67 (2021) 101972. doi:10.1016/j.rcim.2020.101972.

[31] C. C. L. Wang, Y.-S. Leung, Y. Chen, Solid modeling of polyhedral objects by layered depth-normal images on the gpu, Computer-Aided Design 42 (6) (2010) 535 – 544. doi:10.1016/j.cad.2010.02.001.

[32] T.-H. Kwok, Comparing Slicing Technologies for Digital Light Processing Printing, Journal of Computing and Information Science in Engineering 19 (4), 044502 (06 2019). doi:10.1115/1.4043672.

[33] H. Mao, T.-H. Kwok, Y. Chen, C. C. L. Wang, Adaptive slicing based on efficient profile analysis, Computer-Aided Design 107 (2019) 89 – 101. doi:10.1016/j.cad.2018.09.006.