NordDesign 2020
August 11-14, 2020
Kgs. Lyngby, Denmark

Automatic geometry alteration when designing for metal additive manufacturing

Julian Martinsson[1], Olivia Borgue[2], Massimo Panarotto[3], Ola Isaksson[4]

Chalmers University of Technology
[1]julianm@chalmers.se
[2]borgue @chalmers.se
[3]massimo.panarotto@chalmers.se
[4]ola.isaksson@chalmers.se

Abstract

During the last decades, the aerospace industry has been interested in converting some of its production from traditional manufacturing to metal additive manufacturing. This technology is attractive to the aerospace industry due to the low production volumes and the need to minimize product mass that characterize the industry. The transition from traditional to additive manufacturing technologies requires designers to take heed of a new set of design and manufacturing constraints. A well-known limitation for the implementation of additive manufacturing is the need for support structures which increase material consumption and post processing time. The research presented in this paper suggests a novel algorithm for automatically altering the geometry of the product to reduce the need for support structures. The algorithm operates directly on the vertices of the 3D mesh by relocating them to mitigate overhang issues. A proof of concept in the form of a software prototype was developed using the Python programming language to demonstrate the algorithm, and how it can be applied to 3D models. The prototype software utilizes a 3D STL-file as input, and it outputs an altered file of the same format. In the process of developing the software prototype some issues surfaced, the most prominent of which was the inability of the algorithm to handle overhangs parallel to the substrate of the powder bed printer. Such overhangs are also referred to as 0°-overhangs due to their normal vector being parallel to the z-axis, thus resulting in an overhang of 0°. Aside from that the algorithm also struggled with models of high complexity, which in some cases resulted in invalid or malformed STL geometries. The research conducted for this paper indicates that further research and development is required. Future research concerning automatic geometry alteration should focus on how to deal with the 0°-overhangs. It should also involve investigating the possibility to alter the geometry of CAD-models instead of STL-files, as CAD-models can contain more information, such as constraints. Information of that nature could enable the algorithm to take customer requirements into consideration as it alters the geometry, thus improving its ability to maintain product functionality.

Keywords: Additive manufacturing, support structures, geometry alteration, automation

1 Introduction

It is a common practice in the space industry to implement casting for manufacturing of metal components, which has many downsides. The casting process is often characterized by long lead-times and is suited for high volume production. These properties have proven disadvantageous to the space industry which typically is characterized by low production volumes (Dordlofva, Lindwall, Törlind, & others, 2016). Lately there has been an increase in demand of space products due to the advent of new actors competing for market shares. Consequently, there is a need to decrease cost and time to market for space products while proposing innovative, high-performance new designs (Öhrwall Rönnbäck, Isaksson, & others, 2018).

In this regard, additive manufacturing (AM) may provide significant benefits to the space industry. Embracing such a manufacturing strategy has the potential to reduce lead-times and overhead costs. This frees up resources and time that could be used for prototyping, increasing the likelihood of innovation (Lindwall, Dordlofva, Öhrwall Rönnbäck, & others, 2017). Another noteworthy benefit is the potential to reduce component weight, which may have especially large impact on more complex parts that are difficult to create using traditional manufacturing (Allen, 2006). However, in order to take advantage of any of these benefits components need to go through a process of redesign to utilize the inherent properties of AM as much as possible (Borgue, Panarotto, Isaksson, & others, 2018).

Additive manufacturing alleviates some of the constraints of traditional manufacturing, however it replaces them with its own set of limitations. Designing with these limitations in mind requires designers to embrace a new mindset. However, transforming an existing mindset into a new one can be difficult (Borgue, Panarotto, Isaksson, & others, 2018). To tackle this challenge there are two generic strategies available: 1) Design new products with AM in mind. In other words, maintain a "Design for Additive Manufacturing" approach. Or, 2) do not change design methodology. Instead, make products and components compatible with AM during the finishing touches. This might include adding support structures or introducing minor changes to make a product easier to print (Dordlofva, Lindwall, Törlind, & others, 2016). While the second strategy could be more compelling to engineers who lack experience with AM, the first strategy will likely generate better results. The problem is that there is often a lack of experience and tools to make the latter strategy work (Lindwall, Dordlofva, Öhrwall Rönnbäck, & others, 2017).

A common well studied limitation when designing for AM are support structures, the purpose of which is to provide support for overhanging structures and supply a path through which heat can be dissipated (Cloots, Spierings, & Wegener, 2013). Though support structures serve multiple purposes, these structures tend to be superfluous in respect to the desired final product geometry and are therefore removed once the component is printed. This is a laborious process that is often performed manually (Thomas, 2009) (Brackett, Ashcroft, & Hague, 2011). Moreover, once the support structures are removed, the material from these structures cannot easily be reused as it would need to be processed back into powder (Brackett, Ashcroft, & Hague, 2011). In this context, a recurrent strategy is to design AM geometries following design guidelines to avoid support structures. However, these design guidelines become yet another design trade-off that designers must consider while ensuring adequate product functionality. Therefore, this study was performed to answer the following research question:

RQ: How can the geometry of a product be automatically altered to reduce the need for support structures while maintaining product functionality?

In the research presented in this article, a series of algorithms to attain automatic geometry alteration were analysed. The geometry alteration algorithms are intended to be of use in conjunction with powder bed fusion 3D printing technology. Using the results of the analysis a Python script was developed with the capability of altering STL-model geometries automatically. The script was tested using a set of benchmark models and was proven to be useful in some situations, although further development is necessary.

1.1 Background

To support the design and printing processes, several software tools can be utilized. There are tools that attempt to optimize printing orientation of a component to reduce support structures and preserve the quality of specific surfaces. Some tools rotate a 3D-representation of the product into different orientations and evaluate mathematically which orientation is most suitable for AM (Strano, Hao, Everson, & Evans, 2013). Sometimes different orientations are given weight based on how suitable they are (Qie, Jing, Lian, Chen, & Liu, 2018).
Such tools are often used in concert with automatic support structure generation algorithms (Leutenecker-Twelsiek, Klahn, & Meboldt, 2016) (Reiner & Lefebvre, 2016) (Vouga, Höbinger, Wallner, & Pottmann, 2012). Inserting support structures manually can be a tedious and difficult task that likely would lead to waste of materials and suboptimal results. Thus, it is often preferable to have a software analyse the product and perform this step automatically (Strano, Hao, Everson, & Evans, 2013). There is a vast selection of computer-based tools and algorithms for supporting manufacturing processes. However, few of those tools propose an automatic modification of a product geometry to make it easier to manufacture with AM and require less support structures. Some examples of automatic geometry alteration algorithms are the ones proposed by Reiner and Lefebvre (2016) or Vouga et al. (2002) (Leary, Merli, Torti, Mazur, & Brandt, 2014). Learly et al. (2014) and Reiner and Lefebvre (2016) presented studies for sculpting approaches for self-supporting structures; Vouga et al. (2002) on the other side, proposed an algorithm for designing self-supporting free form shapes. However, most strategies require a high level of user input and consume vast computational resources. There is a need for non-resource intensive computational algorithms able to perform automatic geometry alterations to optimize manufacturing and reduce support structures for AM.

2 Method

This project was part of a master thesis, which began with a literature study that was conducted by analysing entries from Google Scholar. The publication list was then complemented with additional entries through backward snowballing (Wohlin, 2014) to reach publications outside the range of Google Scholar. Besides a literature study, two interviews were held. These interviews provided the input of potential end users, as well as compliment the knowledge extracted from the literature study. Both interviews were semi-structured in nature. The first interview was held with an expert in the subject of metal additive manufacturing, and the second with a manager from a company manufacturer of space products. The topics that were discussed during the interviews were:

- Quality issues associated with support structures
- The importance of model orientation
- Potential expectations from a geometry altering software
- How geometry alteration could be of use to space manufacturing companies.

The development of the automatic geometry alteration algorithm was divided into two stages. The first stage consisted of the development of a problem detection algorithm capable of detecting problematic overhang angles and providing information about where in the model these problems exists. The second stage was the development of the geometry alteration algorithm, which utilizes the problem detection information and attempts to improve the product geometry. In both stages the work in progress code was repeatedly tested against the benchmark geometries. A sketch of the workflow can be seen in Figure 1. The geometry benchmark models were exported to the STL-format. The STL-format was chosen as the preferred model format due to its prominent usage in additive manufacturing. Additionally, most CAD tools support importing/exporting STL-files. The benchmark models provide a variety of challenging features and range in model complexity. These benchmarks assisted in finding issues with the explored algorithms, as well as their potential strengths.

Through brainstorming sessions, a set of algorithm alternatives was developed for both the problem detection and the problem correction scripts. When selecting an algorithm for problem detection, a thorough evaluation of each algorithm was conducted based on how well they performed, how easily the overhang angle threshold could be varied, and by its development feasibility. The selection of a geometry alteration algorithm utilized a weighted Pugh matrix (Ulrich & Eppinger, 2012). The criteria for the final script (geometry detection and geometry alteration algorithm combined) were derived from problems that were uncovered during the literature studies. On top of those criteria three others was added:

1. Preserve product functionality after geometry alteration.
2. The script should be able to be run on an average modern computer
3. The script should be realizable during the set time of the master thesis (six months)

The programming language of choice was Python 3 due to its associated vast mathematical library “Numpy” (NumPy developers, 2020), and its open-source characteristics. An additional benefit to using Python was the possibility to pack the entire application (graphical interface and scripts) into a single executable program for the Windows platform.

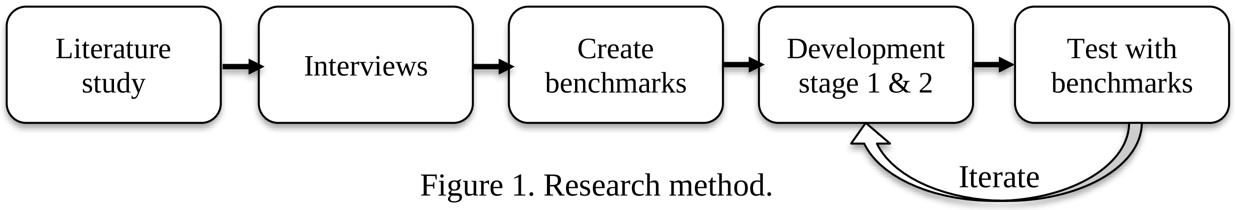

Figure 1. Research method.

3 Results

3.1 Interviews and literature review

Support structures are a problem that concerns both manufacturing companies and academics. This was apparent both from the literature study (Cloots, Spierings, & Wegener, 2013) and the interviews. In the interview with the AM expert it became clear that the tools that already are available are not enough. The expert specifically expressed a dissatisfaction with current orientation optimization algorithms, as they do not account for how hard the support structures will be to remove and might therefor suggest inappropriate orientations. Furthermore, the expert noted that it is often favourable to have any necessary support structures attached to flat surfaces that are easy to access as this makes the manual removal easier. This is also supported by literature (Thomas, 2009) (Brackett, Ashcroft, & Hague, 2011).

Company practitioners stated during the interviews that some surfaces should not be allowed to be altered, nor be connected to any support structures. This ties together with the criterion that the functionality of the final product must remain intact. This is of great importance when designing for example an interface surface that needs to fit together with another part. Any alteration to such a surface might render the interface incapable. Some surfaces should also strictly be avoided when placing support structures. For example, in situations where support structures are required in enclosed or hard to reach areas such as the inside of a curved wave guide. In such cases it would be preferable if an alternative orientation could be found that does not require support structures in delicate locations. This suggests the need for the designer to select important surfaces on the 3D-model which under no circumstances are to be touched, either by support structures or geometry altering algorithms. These statements are also supported by studies (Vayre, Vignat, & Villeneuve, 2012) (Ponche, Kerbrat, Mognol, & Hascoet, 2014).

Using the knowledge obtained from the interviews and the literature study it was possible to initiate the development process.

3.2 Algorithm for problematic geometry detection and alteration

In total, three algorithms were generated for problem detection, and another three for geometry alteration. A sketch for each problem detection algorithm is presented in Figure 2.

3.2.1 Problematic geometry detection algorithm

The following algorithms were generated for detecting problems in an STL-model:

1. ***Voxel parsing***: The STL-model is converted to a voxel-format. An algorithm could then parse through every voxel and ensure that they are all properly supported. This could be done by examining adjacent voxels, ensuring that there is at least one directly or diagonally underneath the subject voxel (Guindon, 2017).
2. ***Machine learning***: A machine learning algorithm could be used to recognize problematic areas of a model using an image recognition approach. Several images of cross sections from the object could be used (Fakhri, Aurélio, & Farida, 2017).
3. ***Normal vector analysis***: An STL-model contains all necessary information to calculate overhang angles. An algorithm could parse through all faces contained in an STL-file and flag faces with problematic overhang angles (All3DP, 2019).

The algorithms were evaluated in detail. The voxel parsing alternative was tested initially by voxelizing different benchmark models. The process of converting STL-files to a voxel matrix with an acceptable resolution proved to be a strenuous and time-consuming process for the CPU, even for small models. High resolutions are often preferable, but there is a significant trade-off between resolution and performance as high-resolution voxel-matrices tend to put a lot of strain on both the CPU and system memory (Singer, 2014). Another drawback with voxel parsing is that while it is relatively straight-forward to identify 45-degree inclines, it is harder and likely more resource-intensive to detect overhangs with angles of variable incline.

The machine learning approach was dismissed because it is by far the most advanced and time-consuming algorithm to develop. To develop a high accuracy image recognition algorithm a large dataset is often required, especially if the classification problem is complex (Beleites, 2013). No such dataset was available during development, and the creation of such a dataset was not possible within the project time frame.

Finally, the normal vector analysis algorithm was successfully implemented into a script. All the necessary information to detect problems using this approach is already integrated into the standardized STL-format, and no conversion into voxels or pictures is required. This results in the normal vector analysis being a relatively fast method, even for models of high complexity.

3.2.2 Automatic geometry alteration algorithm

Geometry alteration algorithms need to be built upon a problem detection approach. However, the different problem detection alternatives have different types of outputs. The voxel parsing script outputs references to unsupported voxel positions, and the normal vector analysis outputs references to STL-format faces that possess problematic overhang angles. These two different kinds of outputs cannot be treated in the same way, and thus the geometry alteration algorithm needs to include what type of problem detection output format it needs to operate. These are the strategies that were generated for geometry alteration:

1. ***Voxel support***: This alternative could be integrated directly into the voxel parsing problem detection process. Once an unsupported voxel is detected, then a supporting voxel is added underneath it (either directly beneath or diagonally).
2. ***Hybrid***: The model is initially searched for issues using the normal vector analysis approach. Once the model has been examined, it is voxelized. Then, a focused voxel support algorithm is run that only targets the problematic volumes of the model. The idea is to reduce the total processing time compared to the first entry in this list (voxel support algorithm) by first detecting any issues using the much faster normal vector analysis, rather than voxel parsing
3. ***Vertex manipulation***: This strategy aims to utilize the STL-format as much as possible to achieve minimum processing times. Since the STL-format contains all faces, normal-vectors and vertices it should be possible to directly alter the positions of the vertices in order to correct any problematic overhang angles. Using normal vector analysis for problem detection should enable a vertex manipulation algorithm to directly identify which vertices should be targeted for manipulation.

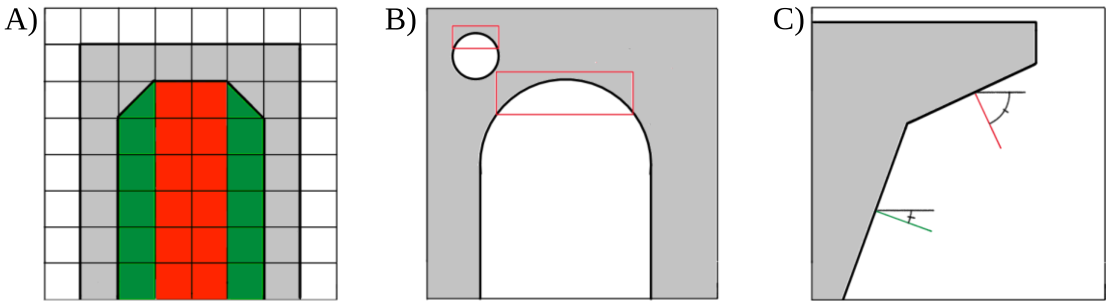

Figure 2. Comparison of strategies for detecting problematic geometries. A) voxel parsing, B) machine learning, C) normal vector analysis.

To evaluate the different geometry alteration strategies a weighted Pugh matrix was utilized (see Table 1). During the evaluation the voxel support alternative is referred to as “VS”, the hybrid as “H” and vertex manipulation as “VM”. The criteria in the matrix are based on how well the different scripts solve the core problem, feasibility and usability. These evaluation criteria were selected based on key insights derived from the literature review and the interviews. Each criterion has its own weight, which represents how important it is. Each strategy is “scored” in each separate criterion with one out of three possible scores: -1 (potentially subpar performance), 0 (unknown or potentially acceptable performance) and 1 (potentially exceeding performance).

In the next sections the strategies will be analysed in further detail with respect to the criteria used in the matrix.

Criterion	Weight (1-3)	VS	H	VM
Ability to solve core problem				
Eliminates problematic angles	3	1	1	1
Minimum overhang angle is variable	2	0	1	1
Maintains product functionality	3	0	0	0
Usability				
Processing speed	2	-1	0	1
Utilizes the STL-format as input and output	3	0	0	1
Feasibility				
Complexity to construct	1	1	-1	-1
Score				
Sum		1	1	3
Weighted sum		2	4	9

Table 1: Scoring of the different geometry alteration algorithms.

Ability to solve the core problem

Problematic overhang angles were the prime issue that needed to be solved, thus the high value of the corresponding criterions weight. At the time of conception these algorithms were all thought to possess the capacity to perform serviceable in this matter. And so, each algorithm received top marks.

The next criterion states that the minimum overhang angle should be variable. Here the Voxel Support alternative scored lower than the others as it is harder to handle adjustments to the minimum allowed overhang angle using the voxel parsing technique. The other algorithms both utilize normal vector analysis, which should handle angle adjustments without issues.

Equally as important as eliminating the problematic angles is the ability to maintain product functionality. However, it is impossible to make a blanket statement for any of the proposed alternatives regarding whether that algorithm will maintain product functionality. For this reason, all alternatives received no points in this requirement as it can only be judged on a case-by-case basis.

Usability

The first requirement in the category of usability treats processing speed. This category is important because a method that takes too long may be too disruptive to the workflow. Parsing through voxels is very time consuming, and thus the voxel support alternative scored the lowest. The hybrid comes next, as it too utilizes voxels but possibly in a more efficient manner.

The next category touches upon interoperability. Most CAD-software can import and export STL-files, which is a widely used format in the world of 3D-printing. Therefore, the algorithm should be able to import and export STL seamlessly to prevent complications to the existing designer workflow. Here the voxel parsing and hybrid algorithms scored lower than vertex manipulation, as they need to at some point convert STL to voxels and back again. This conversion of data structure may cause irredeemable loss of information.

Feasibility

Finally, the issue of complexity is addressed. Voxel parsing appeared to be the easiest alternative to implement. Therefor it scored the highest in this requirement. Vertex manipulation and the hybrid both received a lower score due to their high complexity and knowledge gaps, as it was not entirely clear at this point how these strategies would work.

Vertex manipulation aggregated the overall highest score and was thus selected for further development. Early tests with some of the least complex benchmarks suggested that there was some merit to this solution.

3.3 Implementation of the selected automatic geometry alteration algorithm

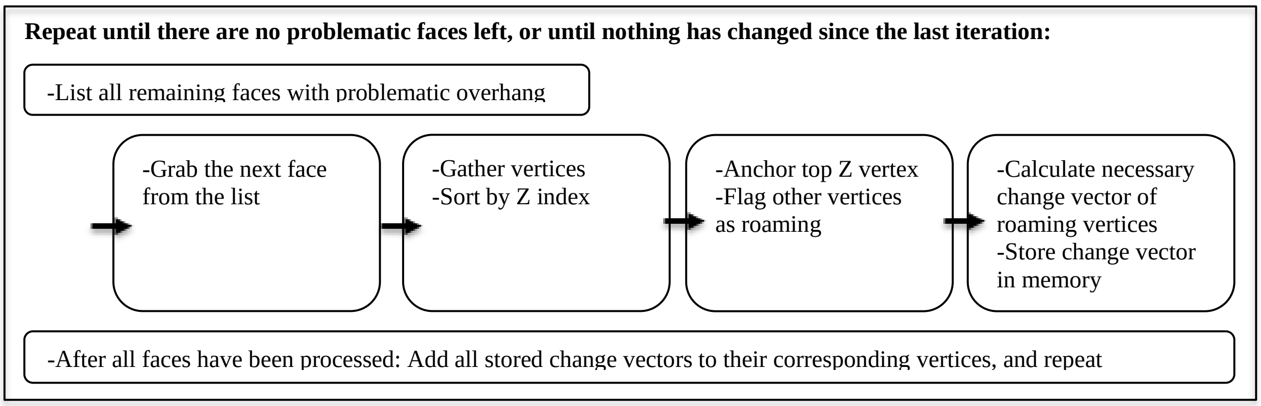
Repeat until there are no problematic faces left, or until nothing has changed since the last iteration:
-List all remaining faces with problematic overhang
-Grab the next face from the list
-Gather vertices -Sort by Z index
-Anchor top Z vertex -Flag other vertices as roaming
-Calculate necessary change vector of roaming vertices -Store change vector in memory
-After all faces have been processed: Add all stored change vectors to their corresponding vertices, and repeat

Figure 3. Diagram of the proposed automatic geometry alteration algorithm.

In order to implement a comprehensive geometry altering algorithm using vertex manipulation it was decided that the algorithm should focus on one face at a time. That way, it would be easier to understand and debug. An algorithm was constructed (see Figure 3), that performs the following set of operations on each face that has been marked by the problem detection algorithm as having a problematic overhang angle:

1. Gather all the vertices of the face and sort them by their respective Z-index (where Z is the printing direction). If they all have the same Z-index, then the overhang angle is 0°, in which case the algorithm stops and moves on to the next face.
2. Flag the vertex with the highest Z-index value as the “anchor” vertex. The anchor vertex will function as the anchor point for this face and will remain stationary. If any of the other vertices share the same Z-index value, then it too will be flagged as an anchor. The remaining vertex (or vertices) will be flagged as “roaming” vertices, as these will be the vertices that move to adjust the angle of the face.
3. For each roaming vertex calculate how far that vertex needs to move for the face to reach a desired minimum overhang angle. The movement that is allowed is strictly along the projection of the original normal vector onto the XY-plane. The reason the original normal vector is used to guide the movement is to attempt to maintain the model’s original shape.
4. Store in memory how far and in which direction the roaming vertices needs to be pushed in order to satisfy the minimum overhang angle for this face. The algorithm has now completed its operations on this face and may move on to the next.

Note that the vertices were never moved during this process. The algorithm merely calculated and stored how the vertices of each face is required to move to satisfy the minimum overhang condition. These changes are stored in vector-format in a stack. In the next step the geometry will be changed based on those stored instructions.

Another loop is initiated, which iterates over each vertex in the model. The loop calculates how each vertex should move individually based on the mean of all stored changes for that specific vertex, and then adds that change vector to the position vector of the vertex. Thus, the changes are in place.

Once all the described operations have been performed, the problem detection algorithm is run again. A new set of faces will be acquired, which are processed through the same set of operations. This process is repeated until either the model stops changing, the problem detection algorithm no longer can find any issues, or the maximum amount of allowed iterations has been reached (to prevent accidental infinite loops). In Figure 4 the results of this algorithm can be seen on one of the benchmark models.

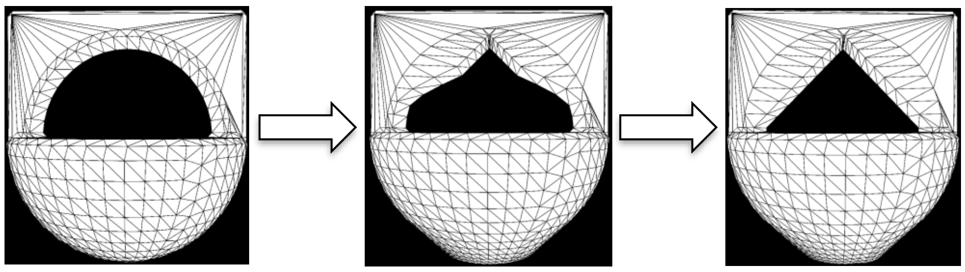

Figure 4. Geometry modification algorithm applied to a benchmark model seen in three stages: Before, in the middle of the process, and after.

A variety of problems were encountered during the development of this method. Faces with an overhang angle of 0° could not be "fixed" using this method. This is because a face that has an overhang angle of 0° has a normal vector that is orthogonal to the substrate. Consequently, the normal vector is also parallel to the Z-axis, thus resulting in its projection onto the XY-plane being the zero-vector. Since the projection of the normal vector is used as guidance for pushing vertices then such faces remain untouched.

Complex curved geometry sometimes resulted in issues where the shape was not preserved as intended. Since this method performs its calculations on one face at a time it does not account for if the movement of the roaming vertices will cause a collision with another face. This may result in invalid geometries that cannot be printed.

4 Discussion

The algorithm that was designed in this project works by pushing vertices away from the model, thus "inflating" it, to resolve problematic angles. This may be the wrong approach for the space industry since every gram need to be considered when launching products into space. Inflating the product would likely result in increased mass. It could be possible to reverse the algorithm, making it "subtractive" instead. Rather than pushing out the vertices with the lowest Z-index value it could pull the vertices with the highest Z-index. However, such a method might impose an elevated risk of creating invalid geometries unfit for print due to face collisions. A second possibility could be to combine these two approaches into a method that neither increases or decreases the mass by both pulling and pushing to decrease the overhang angles.

In its current state the software should not be relied upon to make intelligent decisions regarding a products geometry. However, it may be used as a guide to identify problems, and sometimes it might even help the designer by hinting at a possible "solution". The software could of course be improved upon significantly, but any geometric alteration would likely always need the approval of a designer before going into production.

5 Further development and future research

There are numerous ideas that could be further explored, ranging from basic things such as optimizing performance, to implementing new features. In this section a few of those opportunities will be discussed.

The problem with treating 0°-overhang could possibly be solved by exploring different strategies. Perhaps it would be helpful pausing the program to let the user make a choice about how to resolve such overhang issues, or possibly by scanning the geometry surrounding the overhang and make an “educated” alteration based on the geometric context. This area needs further research, as different situations calls for different solutions. The research effort yielded no generic solution for resolving 0°-overhang issues.

The user experience could also be vastly improved. In the final stages of the project a basic GUI was created to provide a means of interfacing with the script, but it lacked a proper 3D-view of the object that was being manipulated. Such a view could be implemented and could be used to review the alterations. It could also be used to select “strict surfaces” that are not to be touched by support structures or alterations as was revealed during the interviews to possibly be a necessary feature.

A significant upgrade to the software would be to have it operate directly on CAD-files. CAD-files has the potential to contain a lot more information than regular STL-files. Thus, the software could make better and more informed alterations based on for example constraints that are stored in the CAD-file. This could be useful for providing a result that still fits the specifications that the model must meet, and for avoiding collisions. Additionally, if the output remained within the CAD-file format, then it would be significantly easier for a designer to edit the results of the geometric alterations.

6 Conclusion

Automatic geometry alteration works to some extent on low-complexity STL-models. In the attempts made during this project it turned out to be less useful on models of higher complexity with lots of curvature, and on models that has 0° overhangs. It proved hard to create an algorithm that could actively strive to preserve product functionality, but it is possible that this could be improved upon if a similar methodology were to be applied on CAD-files rather than STL-files, as CAD-files has the possibility to contain much more information such as constraints.

The usefulness of this kind of technology in the space industry is unclear. The software could provide guidance to designers and suggest possible alternatives to geometry that normally would not be convenient to print. However, the strategy of pushing vertices results in an increased final volume, which means that the altered product will have more mass. Thus, the current strategy employed by the geometry alteration algorithm might be a bad fit for the space industry. It could be interesting to explore a subtractive strategy, where instead of pushing vertices, they are pulled upon to reduce overhang, or possibly a combination in order to maintain mass.

Source code

The source code is available on Github using the following URL:
https://github.com/johnmartins/am-geoalt

The release (version) of the code that was used to produce the results seen in this paper is *v0.1*:
https://github.com/johnmartins/am-geoalt/releases/tag/0.1
The benchmark model used to produce the results seen in figure 4 can also be found as an attached asset for the *v0.1* release.

Citations and References

All3DP. (2019, 02 14). *STL File Format (3D Printing) – Simply Explained*. Retrieved from All3DP: https://all3dp.com/what-is-stl-file-format-extension-3d-printing/#pointone

Allen, J. (2006). *An investigation into the comparative costs of additive manufacture vs. machine from solid for aero engine parts.* Tech. rep., ROLLS-ROYCE PLC DERBY (UNITED KINGDOM).

Beleites, C. (2013, 03 6). *How large a training set is needed?* Retrieved from stackexchange: https://stats.stackexchange.com/questions/51490/how-large-a-training-set-is-needed

Borgue, O., Panarotto, M., Isaksson, O., & others. (2018). Impact on design when introducing additive manufacturing in space applications. *DS 92: Proceedings of the DESIGN 2018 15th International Design Conference,* (pp. 997-1008).

Brackett, D., Ashcroft, I., & Hague, R. (2011). Topology optimization for additive manufacturing. *Proceedings of the solid freeform fabrication symposium, Austin, TX, 1,* pp. 348-362.

Cloots, M., Spierings, A. B., & Wegener, K. (2013). Assessing new support minimizing strategies for the additive manufacturing technology SLM. *Solid freeform fabrication symposium,* (pp. 631-643).

Dordlofva, C., Lindwall, A., Törlind, P., & others. (2016). Opportunities and challenges for additive manufacturing in space applications. *DS 85-1: Proceedings of NordDesign 2016, Volume 1, Trondheim, Norway, 10th-12th August 2016,* 401-410.

Fakhri, K., Aurélio, C., & Farida, C. (2017). *Image Analysis and Recognition.* Montreal: Springer.

Guindon, M.-A. (2017, 11 26). *What is voxelization*. Retrieved from Quora: https://www.quora.com/What-is-voxelization

Leary, M., Merli, L., Torti, F., Mazur, M., & Brandt, M. (2014). Optimal topology for additive manufacture: A method for enabling additive manufacture of support-free optimal structures. *Materials & Design, 63*, 678-690.

Leutenecker-Twelsiek, B., Klahn, C., & Meboldt, M. (2016). Considering part orientation in design for additive manufacturing. *Procedia CIRP, 50,* 408-413.

Lindwall, A., Dordlofva, C., Öhrwall Rönnbäck, A., & others. (2017). Additive manufacturing and the product development process: Insights from the space industry. *DS 87-5 Proceedings of the 21st International Conference on Engineering Design (ICED 17) Vol 5: Design for X, Design to X, Vancouver, Canada, 21-25.08. 2017,* (pp. 345-354).

NumPy developers. (2020, 02 06). *Numpy*. Retrieved from Numpy: https://numpy.org/

Ponche, R., Kerbrat, O., Mognol, P., & Hascoet, J.-Y. (2014). A novel methodology of design for Additive Manufacturing applied to Additive Laser Manufacturing process. *Robotics and Computer-Integrated Manufacturing, 30,* 389-398.

Qie, L., Jing, S., Lian, R., Chen, Y., & Liu, J. (2018). Quantitative suggestions for build orientation selection. *The International Journal of Advanced Manufacturing Technology, 98*, 1831-1845.

Reiner, T., & Lefebvre, S. (2016). Interactive modeling of support-free shapes for fabrication.

Singer, P. (2014, 08 31). *Handling huge matrices in Python*. Retrieved from Medium: https://medium.com/@ph_singer/handling-huge-matrices-in-python-dff4e31d4417

Strano, G., Hao, L., Everson, R. M., & Evans, K. E. (2013). A new approach to the design and optimisation of support structures in additive manufacturing. *The International Journal of Advanced Manufacturing Technology, 66,* 1247-1254.

The European Powder Metallurgy Association. (2017). *Introduction to additive manufacturing technology, 2nd edition.*

Thomas, D. (2009). *The development of design rules for selective laser melting.* Ph.D. dissertation, University of Wales.

Ulrich, K. T., & Eppinger, S. D. (2012). *Product design and development.* McGraw-Hill/Irwin. Retrieved from http://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=cat07470a&

AN=clc.1d5e579d23b748d58cf5f8ba923cac48&site=eds-live&scope=site&custid=s3911979&authtype=sso&group=main&profile=eds

Wang, D., Yang, Y., Yi, Z., & Su, X. (2013). Research on the fabricating quality optimization of the overhanging surface in SLM process. *The International Journal of Advanced Manufacturing Technology, 65*, 1471-1484.

Vayre, B., Vignat, F., & Villeneuve, F. (2012). Designing for additive manufacturing. *Procedia CIrP, 3*, 632-637.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, (pp. 1-10).

Vouga, E., Höbinger, M., Wallner, J., & Pottmann, H. (2012). Design of self-supporting surfaces. *ACM Transactions on Graphics (TOG), 31*, 1-11.

Öhrwall Rönnbäck, A. B., Isaksson, O., & others. (2018). Product development challenges for space sub-system manufacturers. *DS 92: Proceedings of the DESIGN 2018 15th International Design Conference*, (pp. 1937-1944).