Final Project - Eroding General 3D Parts

Submitting: Michael Kellner, 308154046

Abstract

3D printing as a manufacturing system offers new possibilities for designers. A key feature of it is the ability to print fully assembled mechanisms, while maintaining functionality. Due to the process of 3D printing, in order to create a mechanism some tolerance must be left between the parts, in order for them not to stick together. This tolerance changes from printer to printer. As such, a solution for eroding (inserting) general 3D parts is needed, to prepare these mechanisms to be 3D printed, instead of manufactured in old methods. In this project, I've implemented a general 3D part inserter. I've implemented it via two approaches, Nef polyhedra and Gaussian Maps, which will be discussed and compared. Also, validation of the desired offset, in order to prevent mistakes and enable automation will be discussed and implemented. Finally, a proof of concept for using this solution to print a 3D puzzle, The Gordian Knot, will be presented.

Related Work

Offsetting polyhedra and triangular meshes is a field of active research and solutions. This problem has many applications in various fields of manufacturing and hence its importance. Most of the existing solutions, in commercial software for CAD, such as Solidworks, deal with structures governed by simple mathematical rules. Otherwise the offsetting mechanisms are less likely to work. Su-Jin Kim et al [3] presented an approach for offsetting triangular meshes using multiple normal vectors. This method creates an average normal vector to each vertex, by smartly averaging the normal of the facets that contain the vertex. The main advantage of their method is the speed of computation, due to the simple computation needed. Farouki [4] shows a purely mathematical approach to offsetting simple solids, such as convex polyhedra, which is probably the base of what is done in commercial parametric design software. Many have also chosen to attack this problem via Minkowski sums, as done in this project. Most of the work done try to find efficient ways to compute the complex Minkowski sum. [2] and [4] deal with this subject. The latter one attacks the problem by computing the Minkowski sum of the convex decomposed parts, and later speeds up the union process by approximating it. To the best of my ability I haven't been able to locate a tool or a research project that aims to achieve the goals as set out in this project.

This project also deals with the skeletonizing of 3D parts, in order to check the validity of the desired offset. This subject is extremely researched. This particular problem has application in segmentation, medical imagery, animation and more. Most of the found research presents novel approaches for computing the skeleton. For example, a hierarchical approach was attempted in [5] to provide better grasp of delicate features. The approach is based on maintaining some points during the convergence process, to keep the details. [6] and many others, have suggested algorithms based on Voronoi cells (since the cell boundary are bisectors of the components of the original part). [7] have proposed and approach to calculate the skeleton via penalized distance functions. Their method is supposed to have a very high efficiency. As demonstrated later in this project, computing the skeleton is quite costly, and these articles and more should be considered in order to improve this project in further research.

Eroding a 3D Part

In this part will be presented two solutions for eroding a 3D part by a given offset. Both solutions are based on the same general approach, but differ in the modules used to implement them. Both solutions are using CGAL library modules. As an input, we shall assume that we are given a path to an OFF file, and an offset to erode. OFF files are the common choice for IO in CGAL. Since most of the 3D printed models are found in STL format, a conversion is needed. For this purpose, MeshConv, an open source project for converting meshes, was used[1]. In the following sections both implementation will be presented, and a comparison between them will be held.

Nef Polyhedra Approach

As described above, as an input we are given the model as an OFF file, and an offset to erode. The steps of the eroding algorithm are presented bellow:

---

1 More about MeshConv can be found at http://www.patrickmin.com/meshconv/
1. **Read the mesh file** - CGAL has a built-in module for reading an OFF file into a Polyhedron_3 type. CGAL also implements a module for reading an OFF file to a Nef_polyhedron. This looks redundant since there exists a constructor of Nef_polyhedron from a Polyhedron_3, but precision makes this a necessity. Since Nef polyhedra are based on encapsulating planes, the vertices of each facet must lay exactly on the same plane. This is not enforced by Polyhedron_3 and as such there is a need to project the facet vertices to same plane in a case of need. This is the reason for the module for reading OFF file to Nef. In the project I used this to implement a safe conversion between Polyhedron_3 and Nef_polyhedron.

2. **Compute the compliment of the mesh** - Nef polyhedron is composed of encapsulation volumes with bounding planes. As such, Boolean operations are extremely simple to use. This is why I've used them to compute the compliment of the part. Nef_polyhedron offers a method for finding the compliment. Since the compliment of a bounded part is unbounded, I've also intersected the compliment with a bounding box of the part (with some margins), so it would be bounded.

3. **Dilating the compliment** - In this stage, the compliment is Minkowski summed with an axis parallel cube with a size of the offset given. It is important to emphasize that although the cube is axis parallel, the part does not have to be so. More so, the summed shape (the cube) could easily be replaced, and the only implication should be an increase in compute time. In this stage, the two approaches differ. In this approach the Minkowski sum function for two Nef polyhedron, which is implemented in CGAL, was used to calculate the dilated compliment.

4. **Computing the eroded part** - With the dilated compliment, it is easy to use the Nef polyhedron's compliment method to once again find the compliment (which is the eroded part). Since again compliment is unbound, it is intersected with the original part. This is the final eroded part.

5. **Converting the OFF file to STL** - Since the result is given by CGAL as an OFF file, another conversion is needed. Again, Meshconv was used for this purpose.

The code implementing this is attached to this report. Also, a guide to the code can be found in the appendix. Examples for outputs of the algorithm can be seen in the table below:

<table>
<thead>
<tr>
<th>Convex Decomposition and GM Minkowski Sums Approach</th>
</tr>
</thead>
</table>

As described above, the difference between the two solution is in stage 3, which is dilating the compliment. In this approach stage 3 is computed as follows:

1. **Convex decomposition** - In this approach the compliment is divided into sub parts which are disjoint and convex. This is done using the convex decomposition method available for Nef_polyhedron in CGAL.

2. **Convex subpart dilating** - For each convex subpart the following operations are performed. Firstly, the Gaussian Map (GM) of the subpart is computed. Secondly, the GM of an axis parallel cube is computed. Thirdly, the GMs are overlaid, and due to the properties of the GM, the result is the GM of the Minkowski sum of the subpart and the cube. Finally, the Minkowski sum itself is computed by calculating the convex hull of the vertices corresponding to the facet of the overlaid GM. This Minkowski sum is the dilated subpart.

3. **Summing convex subparts** - As mentioned above, Nef polyhedra are excellent for Boolean operations. As such, each dilated subpart is converted to a Nef polyhedron, which are summed by union to a result. This result is the dilated compliment calculated in stage 3.
Comparison Between the Two Approaches

In order to compare the two approaches, the same models were eroded with both approaches. A summary of the models, their complexity and the run time can be seen below. The runtime is for the eroding step only. The experiments were done on an Ubuntu 64-bit VM, with 4 CPU cores and 6GB of RAM running on i7-7500U processor. As can be seen, the Nef approach is faster than the GM approach. Since most of the stages are identical, it is probably due to the difference in stage 3. From examining the CGAL documentation, it can be seen that the Nef Minkowski sum also uses the approach of convex decomposition and summing convex pairs (similar to what was implemented in the GM approach). Hence, I’ve not been able to point a major difference between the approach to explain the time gap, and it can be examined as a further research.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Vertices, Facets</th>
<th>Nef Runtime</th>
<th>GM Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>E (letter)</td>
<td>24, 20, 84</td>
<td>0.558754 s</td>
<td>1.01433 s</td>
</tr>
<tr>
<td>Blue (Gordian Knot part)</td>
<td>48, 92</td>
<td>1.31536 s</td>
<td>1.96868 s</td>
</tr>
<tr>
<td>Stanford Bunny</td>
<td>148, 292</td>
<td>26.3907 s</td>
<td>125.674 s</td>
</tr>
<tr>
<td>Elephant</td>
<td>2775, 5558</td>
<td>662.822 s</td>
<td>1429.66 s</td>
</tr>
</tbody>
</table>

Validating the Desired Offset

A wanted feature would be to validate the desired offset. The user, which can be wrong, inputs an offset to erode. If this offset is too big, the eroding could cause the eroded part to have different topology than the original one. To prevent this, a feature was added to the eroding algorithm. This feature validates the desired offset as follows:

1. **Calculating the part’s skeleton** - A skeleton, non-formally, is a repeated eroding of the part, until a convergence to a graph is achieved, which in some way describes the original part. CGAL offers a module for computing a mean curvature skeleton. This feature requires the mesh in question to be triangular, and contain many vertices (around thousands of points). The latter one is not a formal requirement, but it was needed for achieving good results. Hence, a triangulation, and isotropic remeshing was performed in order to make the mesh match the requirements. An example for a skeleton calculation can be seen below.

2. **Inflating the skeleton** - To check whether the erosion is possible, we can use the skeleton. Inflating the skeleton is the opposite operation to eroding the part. So, if we inflate (via Minkowski sums) the skeleton, with the same polyhedron to be eroded (in our case, an axis parallel cube), it would be possible to check whether the offset is valid. The inflating of the skeleton is done by iterating the edges of the skeleton, and calculating the Minkowski sum of the edge and the axis parallel cube. Lastly, these boxes are united together, to get the result.

3. **Checking the offset** - Since the inflation is opposite to the erosion, if the inflated skeleton is contained in the original part, the desired offset is valid. Again, Nef polyhedra are the most suitable for containment checks.

Examples of skeletonizing output can be seen below. As seen, the skeleton has a tendency to be round and curvy, even if the model is not. This creates a problem, when the offset is close to the possible limit. Also, the remeshing algorithm requires a wanted maximal edge length. Under the assumption that most models will be of centimeter size, 0.1 nm is a suitable constant for it (and was used), but if larger or smaller models are used, bad results are computed, which leads to the program not enabling the eroding.
3D Printing Proof of Concept

As a possible example from a use of the eroding algorithm, we can consider the printing of puzzles. A puzzle, for the purpose of this article, is a 3D assembly of separate parts, which can be separated (and reassembled of course). These puzzles are commonly used as riddles in various levels. 3D printing offers new opportunities for the world of 3D puzzles. In this article will be shown a proof of concept for printing puzzles on a 3D printer, in their disassembled and assembled form. The puzzle that will be used for this proof of concept is the Gordian Knot puzzle, which consists of six axis parallel parts. The puzzle parts were found in the online repository Thingiverse.

Assembling a Puzzle from 3D Printed Separated Parts

For a first step, we set to show the possibility of printing the parts of Gordian Knot separated, and assembling the puzzle afterwards. Since the puzzle parts, which are usually manufactured by other means, are without the tolerance needed to be manufactured in a 3D printer, an erosion is needed. For this proof of concept, an Ultimaker 3 FDM 3D printer was used, with PLA as the print material. The parts were printed with 0.5 mm space between the parts (in planning). Pictures of the parts and the assembly can be seen below. The assembly process was a bit difficult, and required some manual filling in order to be assembled. This was mostly due to the brim, an addition added by the printer at the start of the print to increase adhesion to the print bed, which made the part a bit bigger than wanted.

Disassembling a 3D Printed Puzzle

The final goal, is to print the puzzle assembled. This has the advantage of not revealing the solution to any one, in contrast to manufacturing the separately and then assembling them. The first try to accomplish this was tried on the Ultimaker described above. Unfortunately, this attempt did not succeed. Since the print was carried out with only one material, support was needed. This support is the reason for the attempt’s failure. As it turns out, some support was impossible to reach (as expected) and did not break under mild physical pressure. It is to conclude that printing with one material on an FDM machine is not suitable for printing an assembled puzzle. The results of this attempt can be seen below. Marked in white is the impossible support simulated by Cura.

Hence, a second attempt was tried on a Connex Objet printer. This printer prints in two materials, and the support be crumbled by mild force, while the main print material does not. To ease the process of removing the support, a power blaster was used. The power blaster blasts the model with highly pressured water, which crumbles the support. The space

---

2https://www.thingiverse.com/thing:11204
after the erosion of the parts was set to $0.8_{\text{mm}}$. The results of this attempt can be seen below. As can be clearly seen the puzzle can be moved successfully. Due to a mistake in assembling when creating the model to print, it was not possible to separate the puzzle completely, but this doesn’t change the proof of concept that was achieved. Also to be noted is that the space that was planned is larger than optimal, which causes the puzzle to move less smoothly (since the parts can be jiggled).

Conclusions and Further Work

This project set out to preset a solution for eroding (inserting) general 3D parts. A general method was described, which relies on dilating the complement of the part. Two approaches were presented to dilating the compliment, one base on Minkowski sum of Nef polyhedra, and the second base on Minkowski sums based on Gaussian maps. Both approaches were tested on several models of different kinds (axis parallel, low poly count, and round) and have generated satisfactory results. Also, both approaches were compared for their runtime, with favorable results for the Nef polyhedra approach. I’ve tried to find the reason for the major difference in runtime, but could not find any suitable explanation. This question can be examined in further research, via isolating each step of the two approaches, and measuring its runtime. Also, diving into the implementation itself for the Nef polyhedra Minkowski sum could lead to some insights.

Secondly, an approach for validating the desired offset to erode was presented and implemented. The current implementation generated satisfactory results for some models. However, it still has some problems. Firstly, some robustness issues have come up while testing, which to the best of my knowledge are related to internal CGAL module of the remeshing algorithm (some assertions fail while computing the remeshing). I’ve not be able to point out the exact source of the problem. Secondly, the method for calculating the skeleton generates curvy skeletons even for axis parallel models. This creates a problem when validating offsets which are close to the maximal possible limit. Lastly, the runtime of the skeletonizing process takes an order of magnitude more time then the insetting itself for small models, and this makes the process less feasible. All these problems need to be furtherly researched in order to make this module viable.

Lastly, a proof of concept for printing 3D puzzles was demonstrated. The Gordian Knot puzzle was printed on an FDM printer with PLA, assembled and disassembled. It was shown that assembly is possible, and that the erosion is indeed needed. It was also shown that printing an assembled puzzle with one material presents a difficulty in removing the support, which is unreachable in some places, and too strong to break. Hence in order to prove the possibility of printing an assembled puzzle, the puzzle was printed on an Object printer, with two materials. The model that was printed was indeed movable, and showed the possibility of printing assembled puzzles. During these attempts, a disadvantage of printing 3D puzzles became apparent. Since the printer prints in one color only, assembling and disassembling these kind of puzzles, which have similarly shaped parts with different colors normally, is more challenging. A step forward in this domain would be to print the puzzles with multicolored materials, so parts could have different colors.
References


6
Appendix

Code Explanation

In the attached solution, there are three main components:

1. **InsetPart.py** - This is a python script which takes as command line input a full path to the STL file to erode, and the desired offset. The script wraps the conversions of MeshConv, and the actual eroding done by the CPP module. The result of the script is an STL file by the same name follow by the desired offset. The script assumes both MeshConv and the compiled CPP module (PartInset) are present in the same directory as it.

2. **MeshConv** - An online open source project for converting between mesh formats. In order to run the script a compatible version of it must to present. Attached is an Ubuntu 64-bit compatible version.

3. **PartInset** - The main CPP module which insets an OFF file. It is run with two command line arguments, a full path to an OFF file, and the desired offset. The file outputs its result as the original filename with the addition of .out. This result is later renamed by the python script. The attached version was compiled for release, on a 64-bit Ubuntu machine.

The CPP module is a compile version of the PartInset project which is also attached. The project has the following main modules:

1. **PartInserter** - The class responsible for executing the eroding for the Nef polyhedron approach. The main function for the class is the inset function. In it can be seen the entire high-level flow presented in the article. This function uses many private inner function to implement each of the steps. PartInserter’s and GMPartInset’s interfaces are identical in order to enable compiling both version with little change as possible to the code.

2. **GMPartInset** - The class responsible for executing the eroding for GM approach. The main function for the class is again the inset function, which is as the high-level flow described above. This function again uses private functions in order to implement this offsetting process.

3. **Skeletoniazer** - The class responsible for validating the desired offset. Its main function is validate_offset, which displays the high-level approach that was presented above. It again uses private functions to implement each high-level step.

4. **Main body** - Responsible for executing the main logic of the solution, and asserting arguments passed. In the attached version, the main body uses the Nef polyhedron approach. In order to compile it to the GM approach, simply change the two uses of the class PartInserter to GMPartInset (with the same template parameters).

The entire module was compiled on an Ubuntu 64-bit VM with CGAL 4.9. Any of the components should print its usage in case of wrong use.