Cylinder-based Approximation of 3D-Objects

Software Lab Project

Students:
Ahmad M. Belbeisi
Cristian Saenz Betancourt
Chaudhry Taimoor Niaz
Benjamin Sundqvist

Supervised by:
Giovanni Filomeno, M.Sc.
Moustafa Alsayed Ahmad, M.Sc.

Technische Universität München

January 2022

Content

Figures

Abstract

A method was developed to approximate any 3D geometry given as triangle meshes by parallel cylinders. Geometries are represented by STL files as an input and converted to a cylindrical approximation as the output. By dividing the geometry into parallel slices with almost constant cross sections, the 3D problem was reduced to several 2D problems where polygons are represented by circles. The main aim of this work is that the approximated result can be as accurate as possible with a small number of cylinders. Some examples are presented, including the approximated geometry of a drivetrain assembly space. A convergence analysis is included. The relevance of this work is oriented to simplify the inside-outside-test for complex geometries.

Keywords: STL, Geometry Approximation, Cylinders.

1. Introduction

1.1. Task description

The first task is to import a 3D geometry from an STL file. This volume should then be approximated by cylinders. All these cylinders need to be parallel. They are defined to be parallel to the y-axis of the given geometry. Furthermore, the shape can be defined as an addition and a subtraction of cylinders. In the following, the added cylinders will be called green, and the subtracted cylinders will be called red. The approximation needs to lie entirely inside the original volume, as this volume should model a construction space for a transmission system. Therefore, it needs to be guaranteed that a point is inside the original volume if it is inside the approximation. The aim is to approximate the shape with as few cylinders as possible while approximating the main features of the geometry well. To evaluate the quality of the approximation, also the volume should be computed and compared to the original volume. The code is tested using multiple different STL-files.

1.2. Motivation

The reason for this cylinder-approximation is to test a method that makes an inside-outside test simpler. Using this approximation, it is easy to determine whether a point lies inside the geometry. If this point lies in any of the green cylinders but in none of the red cylinders, it lies inside the geometry. For the test of each cylinder, only the y-value must be compared to the y-range of that cylinder, as they are parallel to the y-axis. Then, the distance of the point in the x-z-plane to the center of the circle needs to be compared to the radius. In practice, the squared distance will be compared to the squared radius to avoid costly square roots. All in all, the approximation of geometries by cylinders leads to a very fast inside-outside test.

1.3. Literature review

Previous works directly related to the approximation of a geometry by parallel cylinders were not found. However, some indirectly related packing methods were interesting and although

they were not implemented, they contributed to the understanding of the problem and to generate ideas for possible solutions.

Random Circle Packing [1]: The source code fills a rectangle with tangent circles. In a given boundary, the coordinates of centers are allocated successively in a random process. The radius of each circle grows until it encounters another circle or the boundary.

Several attempts were made to improve the code and to make it more adaptable, such as allowing the movement of the centers and the possibility to be applied in any type of polygon (boundary). Nonetheless, it was no longer used in the project because it required a lot of circles and the randomness implied long running times.

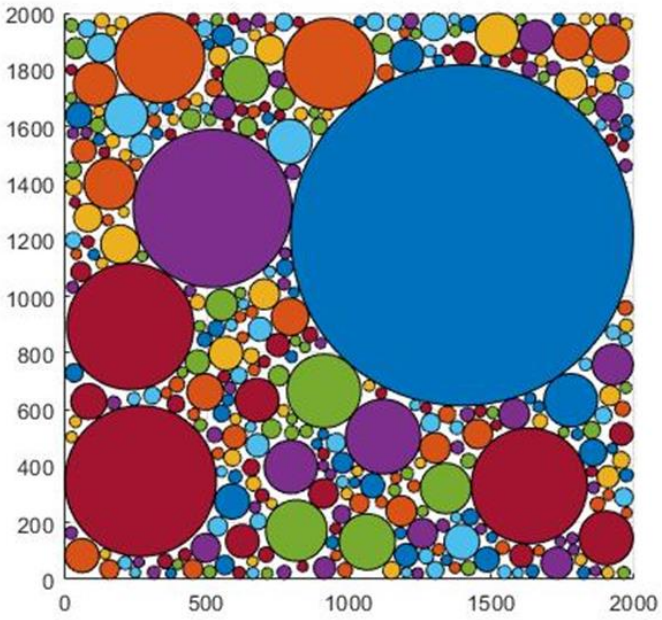

Figure 1: Circle Packing

Collins and Stephenson Circle Packing [2]: It consists of a configuration of circles with a specified pattern of tangencies defined in a given triangulation. This method comprises a system of nonlinear equations to find the radii, resulting in a complex mathematical problem.

The random circle packing and the circle packing by Collins and Stepheson inspired the idea of tackling the 3D problem by solving a set of 2D circle packing problems. Those two methods are mostly focused on tangent circles. By contrast, this work is based on intersecting circles to avoid undefined spaces inside the approximated 3D object.

Another idea was, to approach this topic by formulating a general optimization problem. The objective function would be the area of the approximation, which should be maximized. The design variables would be the number of red and green cylinders and the respective positions and radii. The constraint would be, that the approximation lies entirely inside the given geometry. This problem definition would lead to a nonlinear, even non-smooth, discrete optimization problem. When looking at some literature [3],[4], it became clear, that this forms an extremely difficult optimization problem. Standard solution methods would not be possible for this situation. It would also be very questionable if a global optimum could be found using

this general formulation. Therefore, in the following, a heuristic approach is described, that solves the problem of 3D approximation by cylinders.

2. Algorithm

The main idea of the algorithm is to reduce the approximation of a 3D geometry by cylinders to an approximation of a 2D-polygon by circles. This is possible, because the task is to use only cylinders which are parallel to the y-axis. Therefore, any cut of the cylinder-approximation which is perpendicular to the y-axis can be regarded as a combination of circles. Moreover, these circles resulting from cutting the geometry are constant in a certain range. Therefore, the first step is to approximate a 2D-polygon by circles. This can then be used to define one layer of cylinders.
In the following sections, the algorithm for the cylinder approximation will be described. Several parameters will be specified, which need to be chosen for the algorithm and which influence the performance. An overview of those parameters will be given at the end of this chapter.

2.1. Circle approximation of polygons (2D)

In 2D, the task consists of adding and subtracting circles to approximate a 2D-polygon.

Addition of green circles

Starting from a point on the boundary, the maximum radius of a green circle is found that touches that point and still lies completely inside the domain. This circle may however cross an edge of the 2D-polygon, if this edge is included in the convex hull of the polygon. That is because all parts which exceed the boundary at these edges will later be subtracted by red circles.

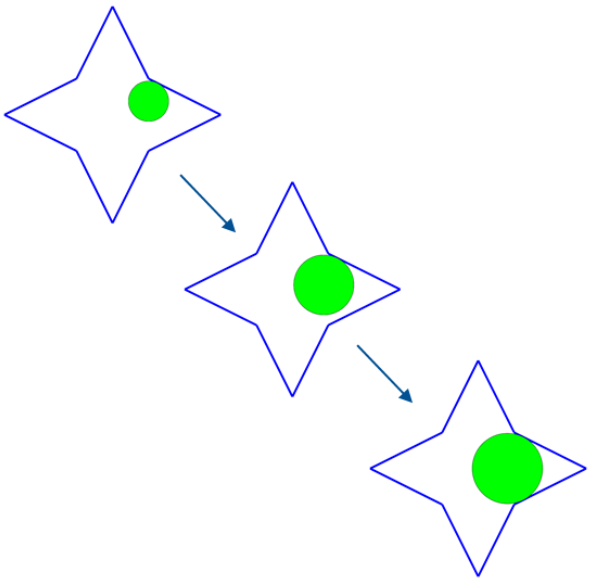

Figure 2: The radius of a green circle is gradually increased until it reaches the boundary. It is defined to always touch a certain point at the boundary.

The initial idea was to always place 5 circles per edge. That means, these 5 circles are defined to be tangential to that edge and then the radius is determined. However, as the number of edges can vary a lot, also the number of circles to approximate the polygon would vary a lot. For that, the number of circles per 2D-polygon was fixed. The number of circles must be chosen as a parameter. The circles are equally distributed around the perimeter of the polygon. For that, first, the points are distributed, where the circles should touch the polygon. Then, starting from each of these equally distributed points, 5 points are generated in the neighborhood. For these points, circles are created as described before. Then, only the circle with the maximum radius remains for each neighborhood.

The maximum radius of each green circle is bounded. It is chosen to be:

$$r_{max} = x_{max} + y_{max} - x_{min} - y_{min}$$

where x and y are the coordinates of the vertices of the polygon. By that, it is ensured that the radius of every green circle is in the correct order of magnitude, so the result does not depend on the scaling of the polygon.

As described before, the circles are defined to be tangential to the edges. This information alone is not sufficient to define a circle with a known radius. A circle with this radius could still lie on either side of the edge. Therefore, it needs to be known, which side is in the interior of the polygon. For that, the convention is used, which assigns a direction to each edge. Then, the inside of the polygon is always at the right-hand side of the edge.

After the tangent point is determined, the maximum possible circle is found using the following algorithm. First, the radius takes some initial value. Then, the radius is iteratively increased or decreased by a certain step-size, depending on whether it fits into the polygon. The step-size starts with half the initial radius and is halved in every iteration-step. By that, the maximum possible radius, such that the circle still fits inside the polygon, is approximated with increasing accuracy for every step.

Subtraction of red circles

First, it is observed that a very large red circle creates a straight edge, if it is subtracted from a green circle. Mathematically, if the radius of the red circles goes to infinity, the edge becomes a perfect line. Using this idea, any edge of the 2D-polygon can be perfectly represented, if an arbitrarily large red circle can be subtracted. This large red circle must not cross any other edges of the 2D-polygon, as it would subtract some parts of the polygon. This condition is fulfilled for any edge, which is included in the convex hull of the polygon. So, all these edges, red circles will be subtracted. Therefore, as mentioned previously, green circles are allowed to overlap the polygon at the edges of the convex hull. The surplus of green circles at these edges will then be subtracted by the red circles.

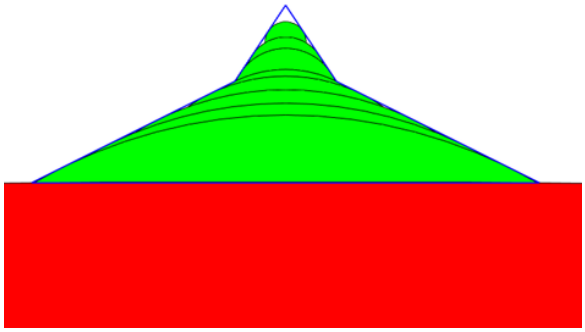

Figure 3: A large red circle (only a small rectangular fraction is visible) is subtracted from some green circles. That leads to a very accurate description of the straight edge.

To implement this subtraction, all edges which are included in the boundary of the convex hull of the polygon are identified. For any of these edges, a large red circle will be defined, which crosses the 2 vertices of that edge and has a center point outside of the geometry. These circles are then subtracted from all previously defined green circles. So, all parts of the green circles, which crossed those edges in the previous step, will be subtracted by the red circles. To ensure that all surplus of the green circles is subtracted, the radii of the red circles are defined to be much larger than the maximum radii of the green circles.

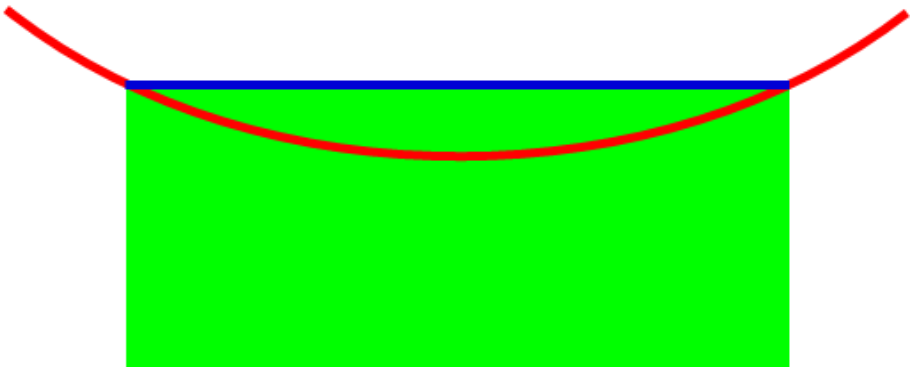

Figure 4: The red circles, that are subtracted from the geometry, are defined by two crossing points and the radius.

The ratio of the radius of the red circles divided by the maximum radius of the green circles is a parameter that must be chosen. In theory, it should be as large as possible to approximate the edge as accurately as possible and to safely subtract all surplus of the green circles. In practice, however, the radius of the red circles should not be too large. As explained in a later chapter, the resulting area of the approximation is computed, by approximating the circles by n-sided regular polygons. If the radii are too large, the n-sided polygons will not be a good approximation of the circles anymore. Therefore, the radius of the red circles needs to be chosen carefully to ensure accurate straight lines as well as an accurate calculation of the area at the end.

Post Processing: Removing green circles

After all green and red circles have been defined, some of the green circles are removed again if they don’t improve the approximation by much. The red circles aren’t removed, as they are

necessary to delete the surplus of some green circles. Also, they lead to good approximations of the edges, so it is unlikely, that removing them would improve the overall result. So only greed circles are deleted, depending on the following 2 criteria.

Firstly, if 2 circles have almost the same center-point, only the one with the larger radius remains. If the distance is smaller than a given value, one circle will be deleted. The critical distance between the center-points is given by the product of the radius and a factor. This factor is also a parameter that needs to be chosen. Later, some cylinders will be reused in the 3D-code, which means that some circles are given for free in the 2D-code. If they are close to a newly created circle and have a larger radius, the new circle will also be deleted.

Secondly, some circles are removed if they do not contribute significantly to the final area. Ideally, all possible combinations of circles would be considered. Then, the combination which uses the smallest number of green circles while the area remains above a certain threshold would be chosen. This strategy is not feasible, as the computation of the area of the resulting shape is one of the computationally most expensive parts of the algorithm. Instead, a heuristic approach is used to remove some circles. First, they are sorted by the radius. Starting with the smallest circle, the area of the approximation without this circle is computed. If the area remains above a certain value and the loss of area in this one step is small enough, the respective circle is deleted. This procedure is repeated once for every circle. In practice, this has resulted in a significant reduction of the number of circles while keeping the computational effort relatively small. This approach introduces 2 new parameters, namely the 2 thresholds for the area. One parameter defines the minimum ratio of the initial area that needs to remain, after some circles are deleted. The second parameter defines the maximum allowable reduction of the area by removing a single circle.

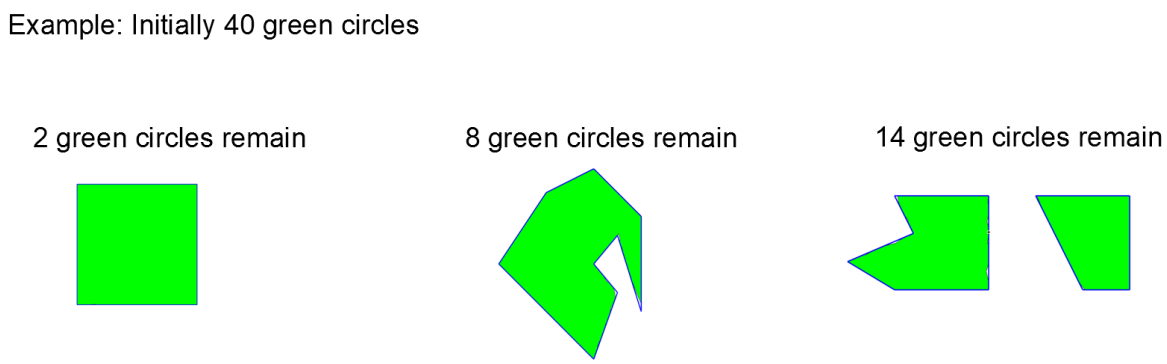

Figure 5: The removal of circles was tested in three small examples. Each started with 40 equally distributed circles around the perimeter. Here, the resulting circle-approximation is shown. A drastic reduction of circles can be observed, while the accuracy is still very high.

Computation of the area

For the post processing-step and the evaluation of the quality of the circle-approximation, the area of the resulting shape needs to be computed. As already mentioned, the computation of the area is one of the most critical aspects for the runtime of the code.

At first, a Monte Carlo algorithm was used. In that approach, some random points are drawn from a uniform distribution inside a bounding-box around the polygon. Then, for each point, it is tested, if that point is a part of the approximated geometry. The area of the polygon can then be estimated by computing the fraction of points inside and outside of the geometry and multiplying this fraction with the area of the bounding box.

This approach could easily be parallelized, which could speed up the process. However, there are many samples necessary which leads to a high effort. Even when choosing a very large number of samples, there will be random fluctuations in the estimation of the area. This makes an accurate approximation of the area infeasible.

Therefore, another approach was chosen to compute the area of the circle-approximation. All circles are approximated by n-sided polygons. Then the polygons which are the unions of all green and red circles respectively are computed. The union of the red circles is then subtracted from the union of the green circles. The area of the resulting polygon is computed. This approximates the area of the circle-shape very well. The accuracy is determined by the number of sides of the regular polygons that approximate the circles. As very large numbers of sides lead to extremely large computation costs, a compromise needs to be found. Here, a number of 600 sides per polygon is chosen. This approach leads to more accurate approximations of the shape than the Monte Carlo approach while the runtime is smaller. Also, the result will be deterministic, as there is no influence on the randomness of the samples.

2.2. Cylinder approximations of triangle-meshes (3D)

Workflow

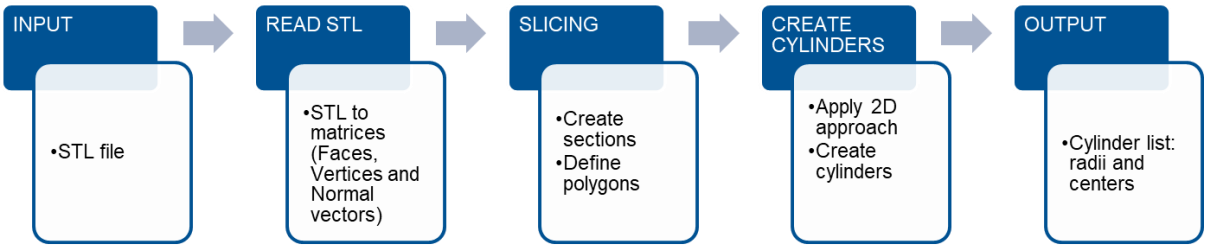

Figure 6: Overview over the workflow for the 3D approximation of objects

Read an STL file

The geometry is read using external functions [5],[6]. The STL file can be binary or ASCII. Three matrices are generated as the output.

[F] = Faces matrix, each row contains the 3 vertices of a face/triangle.
[V] = Vertices matrix, each row contains the x,y,z coordinates of a vertex.
[N] = Normal vector matrix, each row contains the 3 components of the normal vector of a face/triangle.

Choose y-values for cutting

The imported STL file should be approximated using the previously explained circle-approximation of polygons. For that, some suitable 2D-polygons need to be defined first. A list of y-values will be defined, which split the geometry into several sections. In each of these sections, a 2D-polygon is defined which is then approximated by circles. The two y-values at both sides of the section together with the circles for each polygon then define the final cylinders. By that, all cylinders are parallel to the y-axis, as required by the task.

First, the position of the y-values needs to be chosen. For that, the minimum number of sections is given by a parameter. This parameter is used to compute the thickness t of each section if the geometry would be divided uniformly. This thickness t is given by the total length of the geometry l and the prescribed number of sections n as $t = \frac{l}{n}$. This value t is defined to be the maximum thickness of each section.

The y-values will however not be distributed uniformly over the geometry but will be chosen more effectively. If the geometry includes planes, which are perpendicular to the cylinder-axis, the corresponding y-values are suitable positions to cut the geometry. As these planes are parallel to the top and bottom surfaces of the cylinders, the cylinders will fit perfectly to the geometry at these planes. Therefore, all y-values at which the geometry has parallel planes will be included if these planes have a significant area. To determine, whether the area of the parallel planes is significant enough, the area is compared to the total cross-sectional area of the geometry. The related y-value for that cut will only be chosen, if this area-ratio is above a certain limit, which is determined by a parameter.

To compute the total cross-sectional area of the polygon, all triangles in the 3D-mesh are projected to the x-z-plane. Then, the union of these triangles results in a 2D-polygon. The area of this polygon is computed, which yields the cross-sectional area.

After these initial y-values are determined, some more y-values might be included. The additional y-values are computed, such that the maximum distance between 2 y-values is controlled. It is the maximal thickness t as defined earlier.

The minimum and maximum y-value of the geometry are always included as y-values. They will be part of the boundaries of the first and the last sections. At first, however, it is checked whether a valid 2D-polygon can be defined at the ends. It might be possible that no cylinder can be defined that reaches to the leftmost or rightmost side of the geometry while remaining entirely inside the original geometry. This is the case, if there is no face of the original geometry, which lies at the ends of the geometry and is parallel to the x-z-plane. If no valid 2D-polygon can be created at one side, a small part of the geometry is removed there. After that, it is possible to define a valid 2D-polygon at that end, as there now is a parallel plane. The position of such a cut is computed by an offset from the ends. This offset is determined as a fraction of the maximum thickness t of the sections. The fraction is given as a parameter.

Create sections by cutting the geometry

Secondly, the previously defined y-values are used to cut the geometry. The first step in the cutting process consists of a function to cut the STL file only in 2 parts, before and after a given value in the y axis. In the second step, that function is reused in a for-loop to cut the geometry several times.

The geometry in STL format is represented by a set of triangles. Each cutting value can be seen as a cutting plane and each triangle as a plane itself. In the first step, the aim is to define two new sets of matrices in an STL-like format ([F],[V],[N]): *New Green Geometry* which is the geometry before and parallel to the cutting plane, and *New Red Geometry* which is the geometry after the cutting plane.

In a for-loop, all the triangles of the geometry are classified into 3 categories: *Green Triangles,* the triangles before the cutting plane and parallel to the cutting plane, which are saved in the "*New Green Geometry"; Red Triangles,* the triangles after the cutting plane and those are saved in the "*New Red Geometry",* and *Blue Triangles,* the triangles which are intersected with the cutting plane.

Since the *Blue Triangles* are intersected, they have to be redefined with a triangulation process such that the intersection line between the two planes is an edge of two of the new triangles. In the general case, the intersection line crosses the triangle through the two edges, generating a smaller triangle and a quadrilateral. The quadrilateral is divided into two new triangles. The new three resulting triangles are saved either in the *New Green Geometry* or in the *New Red Geometry* according to their location.

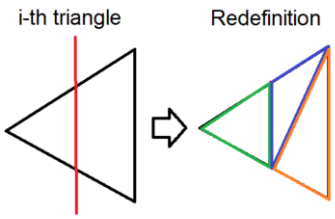

Figure 7: To cut the mesh-geometry into two parts, some triangles need to be redefined.

The previously described function is used as many times as the number of cutting points. Having the first cut performed, the next cutting point always lies in the resultant *New Red Geometry*, therefore the cutting function can be again applied to that geometry and successively until the last cutting point.

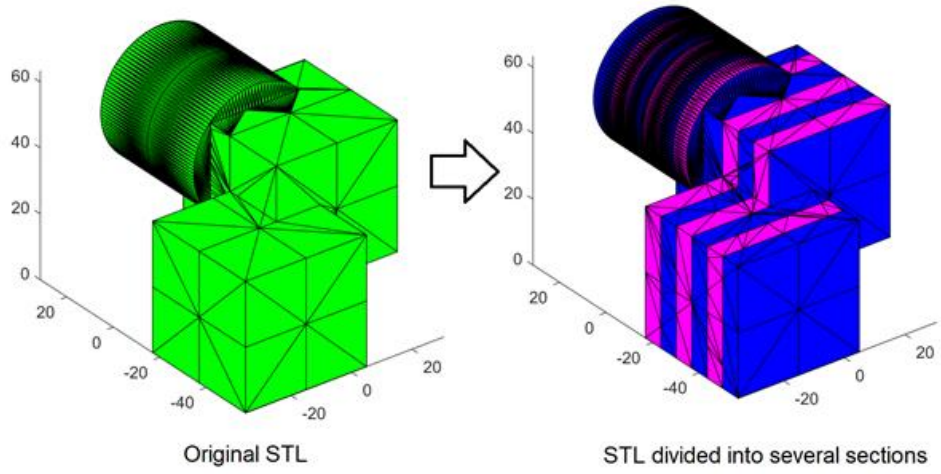

Figure 8: Before and after dividing the geometry into sections.

Define 2D-polygons

Thirdly, the 2D-polygons can be defined from the y-values and the sections in between. The sections are now given as triangle-meshes ([F],[V],[N]). As the approximation should lie completely inside the original geometry, the 2D-polygons should also lie entirely inside the geometry, if they are extruded between the 2 y-values at both sides of the section. Then, the resulting cylinders will also lie completely inside the geometry.

To generate the 2D-polygons for a section, first the 2 polygons at each side of the section are computed. For that, all edges of the triangles which lie at one of the 2 sides are determined. These edges are stored in a list and are combined into 2D-polygons using a graph. The general idea is taken from the code given in [7]. The nodes are determined the following: Whenever two vertices coincide, there will be an edge that connects the respective lines. This can be used to define a graph. By using a depth-first search for each subgraph, the correct order of the lines is determined, such that they form a 2D-polygon. The robustness of this method was improved compared to the original code. After the graph is defined, it is simplified, which means that all multiple connections are removed. Then, all nodes are removed, if they only have one edge, as this situation would not be possible in a well-defined polygon. So at the end, an ordered list of lines in the cutting plane is obtained, which can be used to define the polygon at that plane.

Now, the algorithm to identify whether an edge of a triangle lies at a plane is explained in detail. Only if exactly 2 vertices of the triangles lie at the plane, the edges will be included. If 3 vertices lie at that plane, which means the triangle is parallel to that plane, the edges are not included. This would lead to loops in the resulting graph, so a depth-first search would not be sufficient to determine the resulting polygon. The problem would become much more complicated. Therefore, only edges are included, if exactly 2 vertices of a triangle lie at the plane. So finally, 2 polygons are computed for each section which lies at both ends of this section.

At first glance, two neighboring sections would share one polygon at their interface. However, due to the selection of the y-values, there will likely be triangles that are parallel to the cutting plane. Therefore, the cross section has a jump at this location. In that case, the 2D-polygons for both neighboring sections of that cut are different. As a result, both polygons need to be computed separately.

After the polygons at both sides of a section are computed, they are intersected. Only the intersection of both polygons can lie completely inside this section. However, it is not guaranteed that the intersection will fit into the geometry. Some triangles in the mesh in-between may further reduce the maximum possible polygon. Therefore, all triangles of the remaining section in between are subtracted to get the final 2D-polygon. To be more precise, the projection of these triangles to the x-z-plane will be subtracted from the polygon. In the end, this ensures that the extrusion of the resulting 2D-polygon lies completely inside the geometry.

For this creation of polygons, as just described, some tolerances are necessary to compare floating-point numbers for equality. Resulting from practical experience, these tolerances are all chosen to be 1e-6.

At the end of this step, 2D-polygons and corresponding intervals on the y-axis are given, that approximate the given geometry by piecewise constant cross-sections along the y-axis.

Reduce number of 2D-polygons

Next, the number of 2D-polygons is reduced if some of them are redundant. Some y-values are removed, if the neighboring 2D-polygons change only slightly. A measurement is used, how different 2 adjacent polygons are. It is based on the ratio of the area of the intersection divided by the area of the union of these 2 polygons. If this ratio is 1, the two polygons are equal. A critical ratio is given as a parameter, that determines, when 2 polygons are considered to be equal. Then, the 2 corresponding sections are merged, which means, that the y-value for the cut in-between is deleted.

After the new y-values are chosen, the geometry is now cut again using these new y-values. Also, the 2D-polygons are defined again. In theory, the 2D-polygons of neighboring sections could just be intersected, if the cut between them is removed. This should lead to the same result as computing the sections and 2D-polygons again using the new y-values. However, that leads to less stable results as it is more sensitive to numerical errors. Therefore, the 2D-polygons are computed again using the new y-values.

Create cylinders

With the 2D-polygons for every section, the cylinders can be defined. The circle-approximation is used for the polygon of every section. By the 2 y-values on each side of the section and the definitions of the circles for the polygon, the cylinders can be created. To reduce the number of cylinders, some cylinders could be used not only in one but in several sections. If possible, some of the cylinders from previous sections are reused in the following sections. For that, the length of some cylinders can be increased and so they reach into these neighboring sections. So without additional cylinders, a larger volume of the given geometry can be covered. In

these neighboring sections, some cylinders might be deleted, if the area can be covered by cylinders from the adjacent sections. So, the total number of cylinders to approximate the volume will decrease by reusing cylinders. To make use of this reduction, the 2D-approximation will be computed for all sections without the postprocessing step, so without deleting any circles at first. Then, it is checked, which cylinders can be reused in other sections by increasing their length.

However, there are many possibilities to decide which cylinders to reuse in other sections and which cylinders to delete. It is not feasible to check all these possible combinations of reusing and deleting cylinders. Instead, a heuristic approach is chosen to define the reuse of cylinders. The sections are processed along the y-axis from left to right. For every new section, it is checked, which cylinders of the previous section fit inside the new polygon. If a cylinder fits in the polygon of the next section, it can be reused.

A cylinder usually fits in the new section, if the corresponding circle does not intersect with any of the edges of the polygon. So that is used as a criterion of whether a cylinder fits in the adjacent section. Note, that intersections at the convex hull are not considered, as there will be red cylinders to subtract any surplus at those edges. This intersection-test is not sufficient, though. In some cases, a circle might lie completely outside of the polygon, so there is no intersection, but it obviously does not lie inside the polygon. To take those cases into account, it is checked in addition, whether the center of the circle lies inside the polygon. This condition is a bit too strict, as there might be some cylinders, which fit inside, even if the center is not inside the polygon. That is true in this case, because of the subtraction of red circles. To exactly find out which cylinders could be reused, would be too complex. So instead, the already mentioned 2 conditions are used to determine, whether a circle fits inside the polygon. The circle must not intersect any edge and the center must lie inside the polygon. These 2 criteria are simple and ensure that no parts of the approximation will lie outside of the original geometry.

Now, a set of cylinders is identified which fits inside the new section. These cylinders are reused in the new section. Then the circles, which are related to the reused cylinders, are added to the 2D-problem of the new section. After that, the postprocessing step as described in the 2D-part is applied to potentially reduce the number of circles. Some of the newly created circles are removed. As some cylinders could be reused, the number of circles, which can be removed, will be higher than without the reuse of cylinders. Therefore, the final approximation will consist of less cylinders. This process of reusing helps to reduce the total number of cylinders while keeping the accuracy at a high level.

2.3. Parameter Overview

In the two previous sections, many parameters were mentioned, that influence the result of the cylinder-approximation of 3D-objects. For further improvements to the code, these parameters could be modified. The following table gives an overview of the most important parameters of the code. The names correspond to the variables used in the MATLAB code.

Parameter	Explanation	Useful values
max_number_circles	Number of circles per polygon. It is the maximum, as some circles might fail to be created.	ca. 20-80
red_radius_factor	Radius of the red circles in the 2D-approximation. It is the ratio of the radius of red circles divided by the maximum radius of the green circles.	Depending on n_sides: ca. 10-50
accuracy_factor	Used when removing close circles. Minimum distance between two circles normalized by the radius.	ca. 0.01
min_area_remain	Used when removing circles based on the area. This is the minimum area relative to the total area that has to remain after removing circles.	ca. 0.99-0.9995
max_area_remove	Used when removing circles based on the area. This is the maximum area relative to the total area that may be removed per circle.	ca. 0.00005-0.005
n_sides	Number of sides for the approximation of circles by regular polygons. (Which is used for area computation of the final shape)	ca. 200-600
number_of_sections	Minimum number of sections for the cutting.	ca. 10-100
area_percentage_parallel	Used when choosing the y-values for the cuts. Part of the total cross-sectional area has to lie at a y-plane, such that this y-value is chosen.	ca. 0.005-0.05
ends_offset_fraction	Used when creating the sections. Determines portion of the ends of the geometry, which may be cut away, as no cylinders fit there.	ca. 0.05-0.5
tol, tol_…	Tolerances for the definition of 2D-polygons	ca. 1e-6
maximal_area_difference-ratio	Used when neglecting some sections and the corresponding cuts. It gives a measure on when to keep the cuts.	ca. 0.995

Figure 9: Overview of the parameters inside the code.

3. Results

Initially, the algorithm was used on simple geometries (e.g. cubes, pyramid, torus) to debug the code and ensure the correct performance by approximating sharp edges, holes and curved surfaces.

The intersection of cubes and a cylinder: In this example, the effectiveness of the subtracting process was tested. The straight lines/sharp edges were correctly approximated.

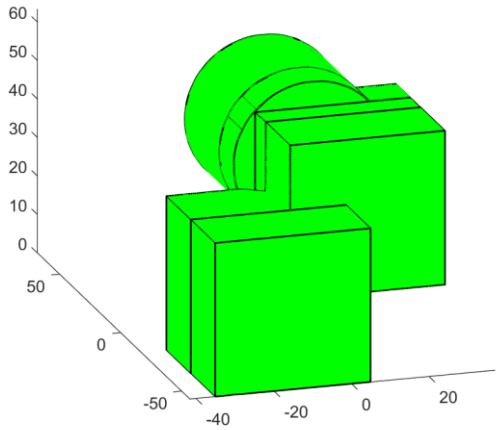

Figure 10: Example geometry: Combined shape of cubes and a cylinder.

Rectangular prism with a hole: With this geometry, the performance of the algorithm in the presence of holes was tested. The output also shows that the function to choose the cutting values works correctly, since those values are located mostly in the region of the hole where the cross section changes considerably.

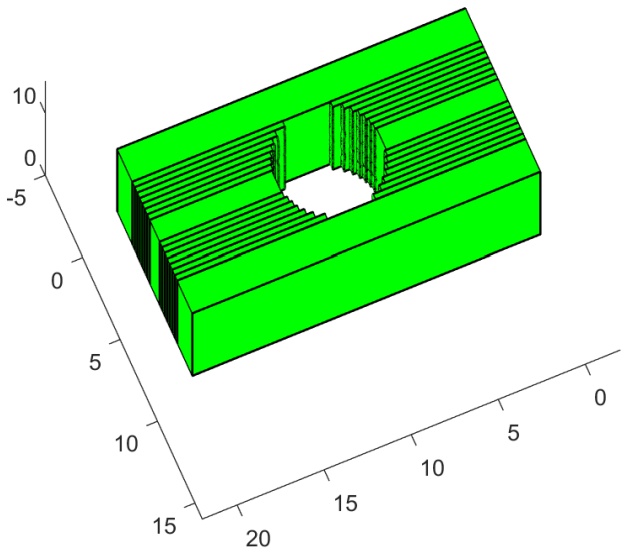

Figure 11: Example object: Rectangle with a hole.

Pyramid: The performance of approximated inclined faces and sharp edges was tested.

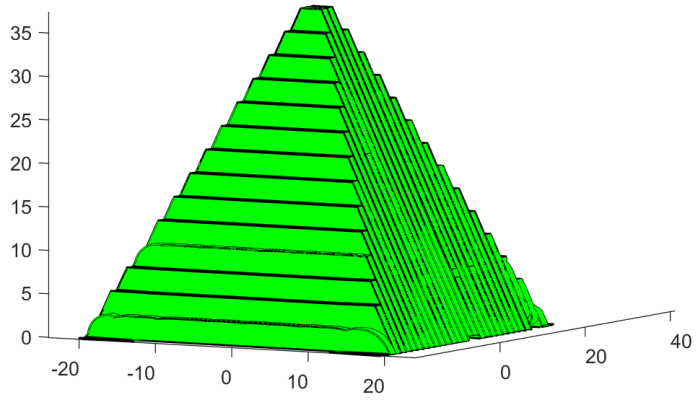

Figure 12: Example object: Pyramid

Torus: The figure shows that curved surfaces can be approximated. When more accuracy is required, the amount of cutting points and cylinder has to be increased.

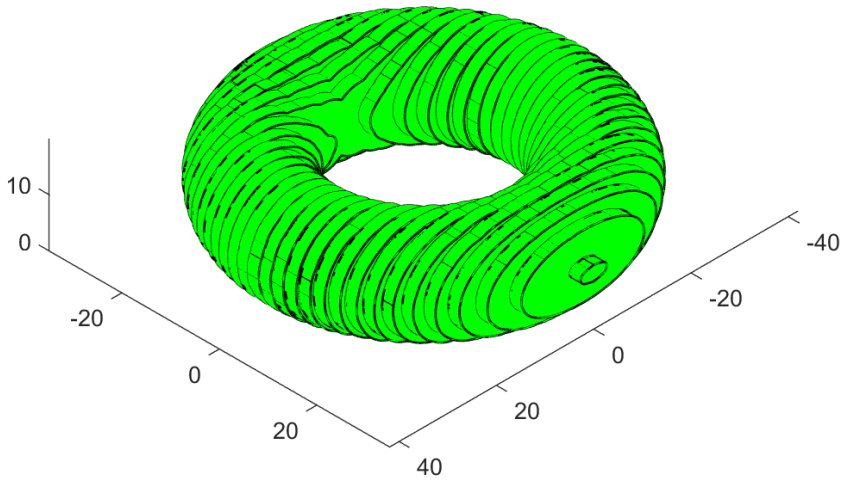

Figure 13: Example object: Torus

Quality of the approximation

Overall, the algorithm can produce good approximations of STL-geometries with a reasonable number of cylinders. However, the computation time is quite high for complex geometries. The quality of the approximation is tested by comparing the volumes of the original with the approximated geometry. As the approximation lies entirely inside the geometry, an increase in the volume means improvement of the result.
The area of the circle-approximations is computed as described earlier. Together with the length of each section, the volume can easily be computed. The volume of the original STL file is computed using a code from [8].
The quality of the approximation depends on all the different parameters that were explained in the previous chapter.

CASE OF STUDY: Drivetrain assembly space (*Bauraum*)

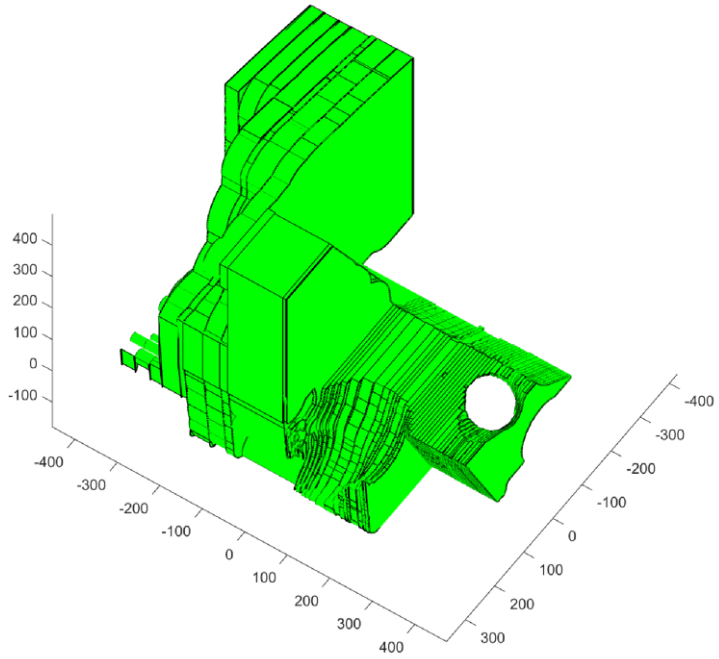

Figure 14: Example object and case of study: Assembly space for a drivetrain.

3.1. Convergence

Two versions of the algorithm were used for further studies based on the drivetrain assembly space (Bauraum)

Algorithm A

- Reuses and removes cylinders
- Computations take longer
- One of the main goals of the project was to implement this algorithm

Algorithm B

- Doesn't reuse nor remove cylinders
- Comparatively faster but uses a higher number of cylinders
- Computations are accurate

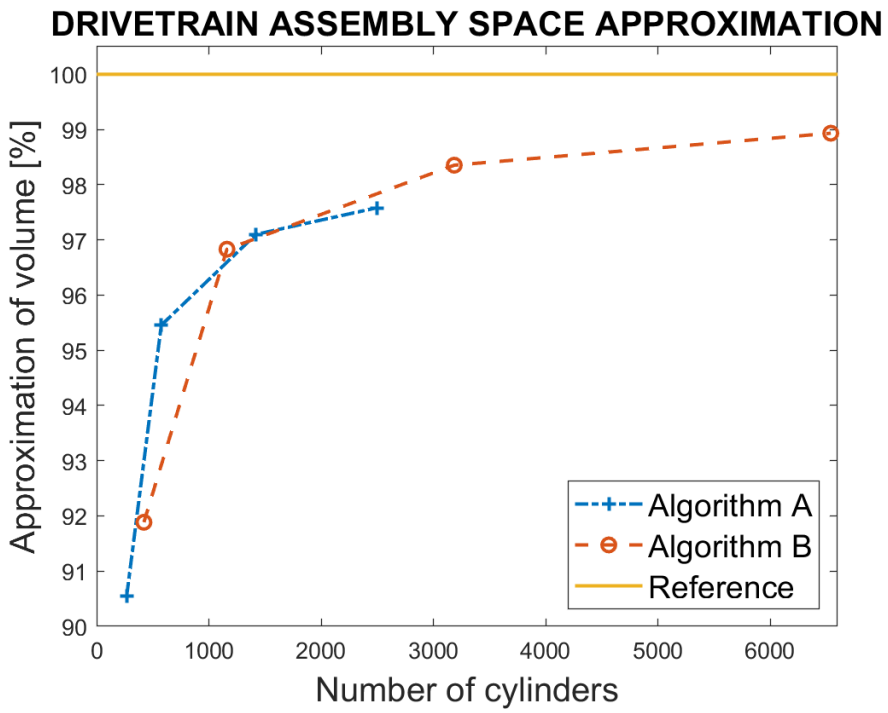

Figure 15: Convergence plot for two different algorithms. The quality is measured by the percentage of approximated volume. The effort is the number of cylinders that is used for that approximation.

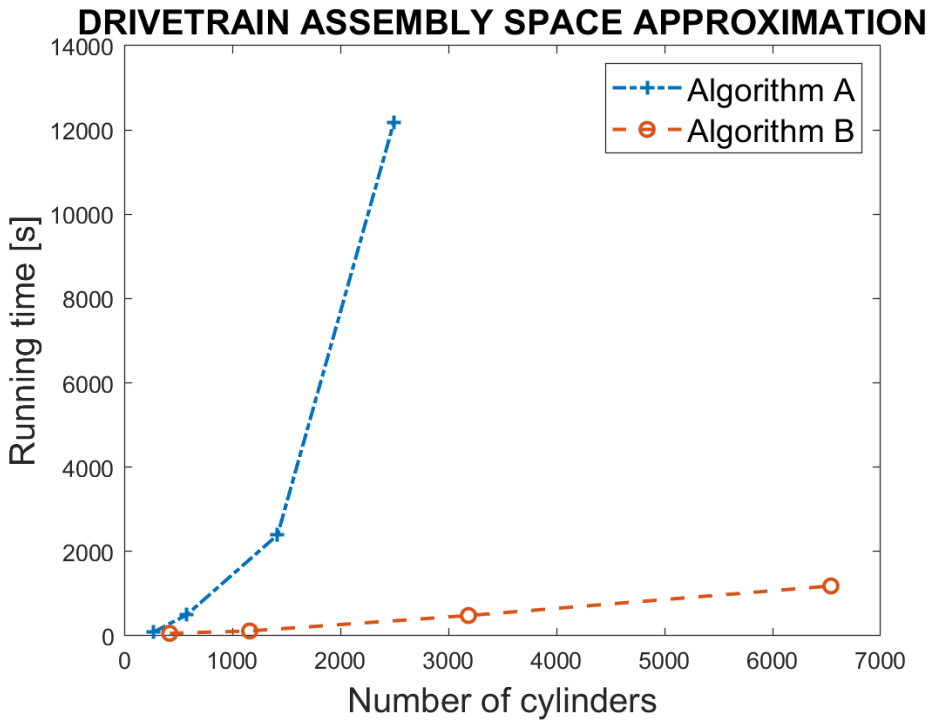

Figure 16: Runtime for two different algorithms. The runtime is compared to the number of cylinders.

3.2. Discussion

If we compare the volume approximation against the number of cylinders for both the algorithms, algorithm B seems to output a more accurate approximation due to the higher number of cylinders. However, algorithm A reaches an approximation of almost 98%, which is

comparable to algorithm B, and is still decent considering it uses significantly fewer cylinders in comparison to algorithm B.

Furthermore, the running time comparison against the number of cylinders for both the algorithms have an increased running time as the number of cylinders increase. However, algorithm A takes significantly more time in comparison to algorithm B. This is due to the fact that there are many iterations in algorithm A, which continuously reuses and removes cylinders, and that repeating step takes a significant portion of the computation time. Meanwhile, algorithm B on the other hand is much faster in comparison due as there are no iterations to remove nor reuse any cylinders.

4. Conclusions

There is sufficient evidence to prove that the final version of the complete algorithm works well with various STL files, which results in a good approximation of only up to a 2% error. The final approximation of the different STL files outputs the required results, and fulfills all of the project objectives, which were to successfully read an STL file, implement an algorithm to identify significant edges in the cross-section area, efficiently remove and reuse cylinders, comparison of volume approximation with the original geometry volume, and lastly to implement it on different STL files. All in all, a combination of different parts of the algorithm accomplishes the desired tasks.

5. Outlook

There are several possible measures that could improve the algorithm of the cylinder-approximation.

As a first idea, better use of the 2D-polygons in the different sections could be made. For example, common edges of neighboring slices could be identified. Then, the same red and green cylinders could be placed at those edges in both sections. By that, the reuse of cylinders in different sections is more effective.

Secondly, the input could be enhanced. If the input wasn’t only an STL-file, but also consisted of any cylinders which are used in the design process, round geometries could be represented better. In the current algorithm, any cylinder of the original model is transformed into a triangle-mesh. By that, corners are introduced. These will in turn be approximated by some new cylinders. In all these steps, information about the original area is lost. So, these new cylinders can never correctly represent the original cylinder and will always cover a smaller area.

Thirdly, the parameters which are needed in the code could be tuned to certain kinds of geometries. Different geometries require different parameters to be approximated most effectively. Therefore, experience and knowledge with some similar geometries would make it possible to choose better parameters.

Lastly, changing the problem definition slightly could also lead to better results.

An idea would be to not only add or subtract cylinders. In addition, intersections of 2 or 3 cylinders could be included as a new possibility to represent the geometry. This would enable a more accurate representation of sharp corners. The inside-outside-test for this new approach would only be slightly more complex.
Moreover, not using parallel cylinders but parallel pieces of cones would also open many more possibilities. That would mean, the radius would not be constant over a certain range of y-values but would vary linearly. So, the test, whether a point lies inside the cone, would not be much more difficult. These varying radii would enable an easier representation of tilted triangles.

6. References

Random circle packing:

[1] Ryan O'Hara (2022). Random Circle Packing in a Rectangle with DXF Output (https://www.mathworks.com/matlabcentral/fileexchange/71088-random-circle-packing-in-a-rectangle-with-dxf-output), MATLAB Central File Exchange. Retrieved January 8, 2022.

Collins and Stephenson Circle Packing:

[2] Collins, C. R., & Stephenson, K. (2003). A circle packing algorithm. Computational Geometry, 25(3), 233–256. (https://doi.org/10.1016/S0925-7721(02)00099-8).

Books on optimization problems:

[3] Bagirov, A. M. & Gaudioso, M. & Karmitsa, N. & Mäkelä, M. M. & Taheri, S. (2020). Numerical Nonsmooth Optimization. (https://doi.org/10.1007/978-3-030-34910-3).

[4] Xue, D. (2020). Solving Optimization Problems with MATLAB®. Berlin/Boston: Tsinghua University Press Limited and Walter de Gruyter GmbH.

Codes to read/write STL:

[5] Pau Micó (2022). stlTools (https://www.mathworks.com/matlabcentral/fileexchange/51200-stltools), MATLAB Central File Exchange. Retrieved January 9, 2022.

[6] Eric Johnson (2022). STL File Reader (https://www.mathworks.com/matlabcentral/fileexchange/22409-stl-file-reader), MATLAB Central File Exchange. Retrieved January 9, 2022.

Code to define-2D-polygon:

[7] Sunil Bhandari (2022). slice_stl_create_path (triangles,slice_height) (https://www.mathworks.com/matlabcentral/fileexchange/62113-slice_stl_create_path-triangles-slice_height), MATLAB Central File Exchange. Retrieved January 9, 2022.

Code for stl-volume:

[8] Krishnan Suresh (2022). Volume of a surface triangulation (https://www.mathworks.com/matlabcentral/fileexchange/26982-volume-of-a-surface-triangulation), MATLAB Central File Exchange. Retrieved January 9, 2022.