[**Task description**](#_p23l6rrb65w7) **2**

[**Algorithm**](#_jj0wmoq7cd68) **2**

[Circle approximation of polygons (2D)](#_f0moe0mtt1wr) 2

[Cylinder approximations of triangle-meshes (3D)](#_rmng6jc95oyb) 5

[**Results**](#_qqa9yz5nqqgd) **7**

[Convergence](#_h2igjbef31cw) 8

[Discussion](#_p5py4tl9v6xp) 8

[**Conclusion and outlook**](#_qrkpjc4q8xi3) **8**

[**References**](#_ev4o2bndwi19) **8**

Cylinder-based approximation of 3D-objects

# Task description

The first task is to import a 3D-geometry from an stl-file. This volume should then be approximated by cylinders. All of 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. So 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 also tested using multiple different stl-files.

The reason for this cylinder-approximation is to test a method which 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 has to 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 origin of the circle needs to be compared to the radius. In fact, the squared distance will be compared to the squared radius as roots can be avoided. All in all, the approximation of geometries by cylinders leads to a very fast inside-outside-test.

# Algorithm

The main idea of the algorithm is to reduce the approximation of a 3D geometry (triangle-mesh/stl-file) by cylinders to an approximation of a 2D-polygon by circles. This is possible, because the task is to use only parallel cylinders. Therefore, any cut of the cylinder-approximation which is perpendicular to the y-axis/cylinder-axis can be regarded as a combination of circles. Moreover, these circles resulting from the cut 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.

## Circle approximation of polygons (2D)

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

**Addition of circles**

Starting from a point on the boundary, the maximum radius of a 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.

The initial idea was to place 5 circles per edge. That means, these 5 circles are defined to be tangential to that edge and then the size is found.

However, as the number of edges can vary a lot, the number of circles per 2D-polygon was then fixed. 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 in each neighborhood. It needs to be chosen, how to distribute these 5 points, which deliver the candidates for the green circles.

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, 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 both sides of the edge. Therefore, it needs to be known, where the inside of the polygon is. For that, the convention is used, that if an edge is defined with a direction from point A to point B, 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 is found, such that the circle still fits inside the polygon.

**Subtraction of circles**

First it is observed that a very large red circle, which is subtracted from a green circle leads to a straight edge. 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 is possible for any edge, which is included in the convex hull of the polygon. Therefore, as explained previously, green circles are allowed to overlap the polygon at edges on the convex hull. The surplus of green circles at these edges will then be subtracted by the red circles.

To implement this procedure, 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 centerpoint outside of the geometry. These circles are then subtracted from all previously defined green circles. So all parts of 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.

The ratio of the maximum radius of the red circles divided by the maximum radius of the green circles is a parameter that has to be chosen. In theory, it should be as large as possible in order to approximate the edge as accurately as possible and to savely 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 any more.

**Postprocessing: 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 in order to delete the surplus of some green circles and because they lead to good approximations of the edges. For the deletion of green circles, 2 criteria are defined.

Firstly, if 2 circles have almost the same center-point, only the one with the larger radius remains. The tolerance of how small the distance between the center-points need to be is 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, only the combination which uses the smallest number of green circles while the area remains above a certain threshold would be used. This strategy is not feasible, as the computation of the area 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, which influence the final result, 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 area when a single circle is removed.

**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. 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 in order 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. Then, the union of the red circles is 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 then the Monte Carlo approach while the runtime is smaller. Also the result will be deterministic, as there is no influence of the randomness of the samples.

## Cylinder approximations of triangle-meshes (3D)

With the aim to use the previously explained circle-approximation of polygons, some suitable 2D-polygons need to be defined. A list of y-values (y-axis = cylinder-axis) will be defined, which split (or cut) 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.

**Choose y-values for cuts**

First, the position of the y-values needs to be defined. For that, a parameter is chosen that gives the minimum number of sections. This parameter is used to compute the thickness of each section if the geometry would be divided uniformly. This thickness is t=total\_length/number\_of\_sections. This thickness is defined to be the maximum thickness of each section. So the minimum number of sections is a parameter that influences the accuracy of the final result.

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 the area of that plane is significant enough. 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 left 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 at the ends is determined by a parameter. The offset of the ends is determined as a given fraction of the maximum thickness of each section.

**Cut the geometry**

Secondly, the y-values are used to cut the geometry. CREATE 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. 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 the sections, first the 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 to 2D-polygons using a graph. The general idea is taken from the code given in [SOURCE!]. 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 is not possible in a well-defined polygon.

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 lie at both ends of this section.

At first glance, two neighboring sections should share one of these polygons. However, due to the selection of the y-values, there will likely be triangles which 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 the section are computed, they are intersected. Only the intersection of both polygons can lie completely inside the geometry. 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. At the end, this ensures that the extrusion of the resulting 2D-polygon lies completely inside the geometry.

**Reduce number of 2D-polygons**

Finally, the number of 2D-polygons is reduced again, if they are redundant. Some y-values are removed, if the neighboring 2D-polygons change only slightly. The measurement, how different 2 polygons are, is based on the ratio of the area of the intersection divided by the area of the union of these 2 polygons. The critical ratio is given as a parameter.

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 and it’s more sensitive to numerical errors. Therefore, the 2D-polygons are computed again from the start.

**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. The length of some cylinders can be increased and so they can be used for neighboring sections without increasing the number of cylinders. 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. 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 which cylinders to reuse in other sections and which cylinders to delete. It is not feasible to check all of these possible combinations of reusing and deleting cylinders. So a heuristic approach is chosen to define the reuse of cylinders. The sections are processed from left to right. For every new section, it is checked, which cylinders of the previous section fit inside the new polygon.

A cylinder normally fits in the new section, if the corresponding circle does not intersect with any of the edges of the polygon (not considering the edges at the convex hull, as there will be red cylinders to subtract any surplus). In rare cases however, a circle might lie completely outside of the polygon, so there is no intersection but it still does not fit. To prevent those cases, it is checked in addition, whether the center of the circle lies inside the polygon. There might be some cylinders, which could be reused even if the center is not inside the polygon, because of the subtraction of red circles. To exactly find out which cylinders could be removed, would be too complex. So the test if the center is inside is used as criterion, as it is simple and ensures 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 reduction of circles as described in the previous chapter is applied. 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 approximation will consist of less cylinders. This process helps to reduce the total number of cylinders while keeping the accuracy at a high level.

# Results

Overall, the algorithm can produce good approximations of stl-geometries with a reasonable amount 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 (SOURCE!).

The quality of the approximation is dependent on all different parameters that were explained in the previous chapters.

## Convergence

## Discussion

# Conclusion and outlook

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

As a first idea, better use of the different 2D-polygons 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 of these steps, information of 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.

# References

Codes to read/write STL:

<https://de.mathworks.com/matlabcentral/fileexchange/51200-stltools>

<https://de.mathworks.com/matlabcentral/fileexchange/22409-stl-file-reader>

Code to define-2D-polygon:

<https://de.mathworks.com/matlabcentral/fileexchange/62113-slice_stl_create_path-triangles-slice_height>

Code for stl-volume:

<https://de.mathworks.com/matlabcentral/fileexchange/26982-volume-of-a-surface-triangulation>