Turkish Journal of Electrical Engineering and Computer Sciences

Volume 32 | Number 6 Article 4

11-25-2024

A Surface-Based Approach for 3D Approximate Convex Decomposition

ONAT ZEYBEK KUŞKONMAZ

YUSUF SAHİLLİOĞLU

Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2024) 32: 774 – 789
© TÜBİTAK
doi:10.55730/1300-0632.4102

A surface-based approach for 3D approximate convex decomposition

Onat Zeybek KUŞKONMAZ, **Yusuf SAHİLLİOĞLU**[*]
Department of Computer Engineering, Middle East Technical University, Ankara, Turkiye

Received: 07.12.2023 • **Accepted/Published Online:** 26.09.2024 • **Final Version:** 25.11.2024

Abstract: Approximate convex decomposition simplifies complex shapes into manageable convex components. In this work, we propose a novel surface-based method that achieves efficient computation times and sufficiently convex results while avoiding overapproximation of the input model. We start approximation using mesh simplification. Then we iterate over the surface polygons of the mesh and divide them into convex groups. We utilize planar and angular equations to determine suitable neighboring polygons for inclusion in forming convex groups. To ensure our method outputs a sufficient result for a wide range of input shapes, we run multiple iterations of our algorithm using varying planar thresholds and mesh simplification levels. For each level of simplification, we find the planar threshold that leads to the decomposition with the least number of pieces while remaining under a certain concavity threshold. Subsequently, we find the simplification level that houses the decomposition with the least concavity, and output that decomposition as our result. We demonstrated experiment results that show the stability of our method as well as compared our work to two convex decomposition algorithms, providing discussion on the shortcomings and advantages of the proposed method. Notably, our main advantage turns out to be on time efficiency as we produce output faster than our competitors which, however, outperform our results for some models from an accuracy perspective.

Key words: 3D approximate convex decomposition, mesh segmentation, shape abstraction, computer graphics

1. Introduction

Shape segmentation is the process of taking a 2D polygon or a 3D polyhedron, and systematically breaking it down into multiple pieces. This is typically carried out in a semantic way where each piece is separated meaningfully, such as separating the handle and spout from a teapot model. Such shape segmentation may be required in many different areas ranging from medical imaging to autonomous vehicles.

Approximate convex shape segmentation, or approximate convex decomposition, is the process of segmenting a 2D or 3D concave shape into smaller, convex parts. It is, in other words, a variant of shape segmentation where a concave mesh is divided into smaller pieces that are all convex and combined to form an approximate of the original mesh. This is performed because there are many applications where the computation being done requires convex shapes to proceed, or convex shapes make such computations much more efficient. For example, video games and simulations make constant use of physics engines for efficient collision detection queries. However, most engines cannot use concave objects in rigid body calculations, meaning concave objects cannot be dynamic physics objects without taking their convex hull or decomposing them into convex parts. Another example is in the area of 3D printing, where printing out convex segments that are assembled by hand is much more material-efficient than having to print concave pieces that may need extra support structures

[*]Correspondence: ys@ceng.metu.edu.tr

printed alongside them. Other examples of the applications of convex decomposition include skeleton extraction and motion planning [17]. Segments that are not necessarily convex are also commonly used in computer graphics for modeling [18], matching [19], and guarding [23] purposes.

The scope of this work is to analyze triangle meshes embedded in 3D and provide a simple method to find an approximate convex decomposition by only taking its surface polygons into account in an iterative manner. We then try to improve our result by automatically and systematically tweaking our parameters and finding the best-performing decomposition.

The majority of convex decomposition algorithms are subtractive and top-down, meaning they start with the overall shape, and repeatedly split it into smaller pieces until all pieces are convex. This often requires complex cutting plane computations and in some cases only works on closed meshes. Other methods are additive and bottom-up, meaning they start from small pieces of the mesh such as polygons or vertices, and combine them until no larger convex pieces can be formed. These methods are either not surface-based, meaning they combine volumes or tetrahedra instead of polygons, or they do not produce good results in a quick manner.

With this work, we provide an additive method for the convex decomposition of 3D meshes that is:

- Surface-based, with a focus on locality and simplicity. In other words, we take into account only the surface polygons and iterate over them based on adjacency, without having to compute overall costs for the entire shape, similar to a greedy approach.
- Self-improves its parameters by running the decomposition algorithm repeatedly with different inputs in order to find the best-performing result.
- Relatively fast at providing sufficiently accurate convex decomposition results as shown in Section 4.

Section 2 presents a literature overview and Section 3 provides our algorithm whose results are discussed in Section 4. Stating our limitations in Section 5, we conclude along with future work directions in Section 6.

2. Related work

The problem of convex decomposition boils down to the problem of finding a viable concavity metric in most cases. Hierarchical approximate convex decomposition (HACD) [1] uses the distance between a point and its projection to the convex hull of the surface it resides on as a basis of its concavity metric, before merging surfaces by considering the two with the least combined concavity. In our concavity calculations, we have used a similar method as described in subsection 3.4.3 and compared our results to that of HACD in subsection 4.3.1. However, our method is more localized, forming groups by iterating over the polygons by adjacency as opposed to calculating the cost of each possible merge option, which in turn leads to faster execution times.

Similar to the method in [22], the approach in [9] employs the merging logic from HACD to form convex hulls for collision detection [21] in complex virtual environments, such as those found in video games, using a game semantics-based concavity metric. In contrast, a previously proposed method [2] uses the volume difference between the mesh and its convex hull as the concavity metric. It tetrahedralizes the input mesh into the leaves of a tree structure and merges the tetrahedra hierarchically.

Similar to our method, the approach described in [10] also iterates over the surface polygons of the mesh and combines them; however, it does this with the goal of finding primitives such as planes, not a convex decomposition. A study more closely aligned with ours is [11], where the triangles of a mesh are iterated to form convex groups using a queue, followed by a final merging step that differs in implementation. However, this method performs scalar triple products to decide the concavity of a surface and the results are significantly overdecomposed. In comparison, our method generally produces fewer convex pieces that capture the original geometry more accurately.

There are also learning-based methods that utilize neural networks. For instance, the approach described in [3] trains on the input shape and reconstructs it using primitive convex shapes. In contrast, the method described in [4] is similar to [3], but while [3] focuses on smooth shapes, [4] generates sharper shapes with lower resolutions by using binary space partitioning (BSP) trees to derive their primitives. Being a completely rule-based method based on geometric optimization, we do not suffer from the poor generalizability of these learning-based methods and also avoid labor-intensive training efforts specific to each object class.

Certain methods split the input mesh until a concavity threshold is met. A simple example is presented in [12] where a 2D polygon is recursively split into two by drawing a line from a concave vector to its optimal visible vertex. Visibility-based decomposition is also related to finding the kernel of a given polygon or polyhedron [20]. The method described in [6] calculates its concavity metric based on the concept of filling a 2D polygon with air and inflating it to its convex hull before iteratively splitting the polygon through its most concave feature. The approach presented in [7] extends this concept to 3D, using the distance between a feature and its convex hull to find the most concave feature and cut the mesh through it using 3D planes. In [8], this method is advanced further by measuring the change in concavity that each cut will produce and selecting features to divide the mesh accordingly for higher-quality decompositions. Another approach, as described in [1], calculates its concavity based on the 2D shape formed by the projection of the Euclidean and geodesic shortest paths between two points and segments the mesh down to convexity using this metric.

Volumetric hierarchical approximate convex decomposition (V-HACD) [14] compares the volume of the convex hull of a mesh with its own volume to calculate the concavity, similar to the approach described in [2]. It then voxelizes the mesh and decomposes it using axis-aligned planes until each piece is sufficiently convex. We use this method as the second comparison method in our test suite. A recent study by [15] presents a similar method that does not voxelize the mesh and employs a multistep search to consider future cuts while finding cutting planes. The method presented in [5] also employs a volume-based concavity metric and utilizes 3D planes to cut the mesh, extending the algorithm to animated meshes. By proposing a surface-based method, we eliminate the additional distortions that these volumetric methods introduce during the switch from the original surface representation to an alternate one.

3. Methodology

The fundamental premise of this algorithm is to iterate over each polygon, using planar and angular equations to determine whether a neighboring polygon is sufficiently convex to be added to the currently forming convex group or not. This process is repeated with different threshold values for the planar equations, as well as varying degrees of mesh simplification applied to the input model. Figure 1 depicts two critical aspects of our algorithm: threshold selection and resolution selection. The former is utilized to determine the optimal parameters, while the latter is employed to adjust the mesh resolution for the best result. The decomposition branch with the best-performing planar threshold within each simplification subtree is selected (Figure 1-left). Subsequently, we select the branch that leads to the simplification with the lowest concavity cost (Figure 1-right). This ultimately results in the final decomposition, which divides the original mesh into convex meshes.

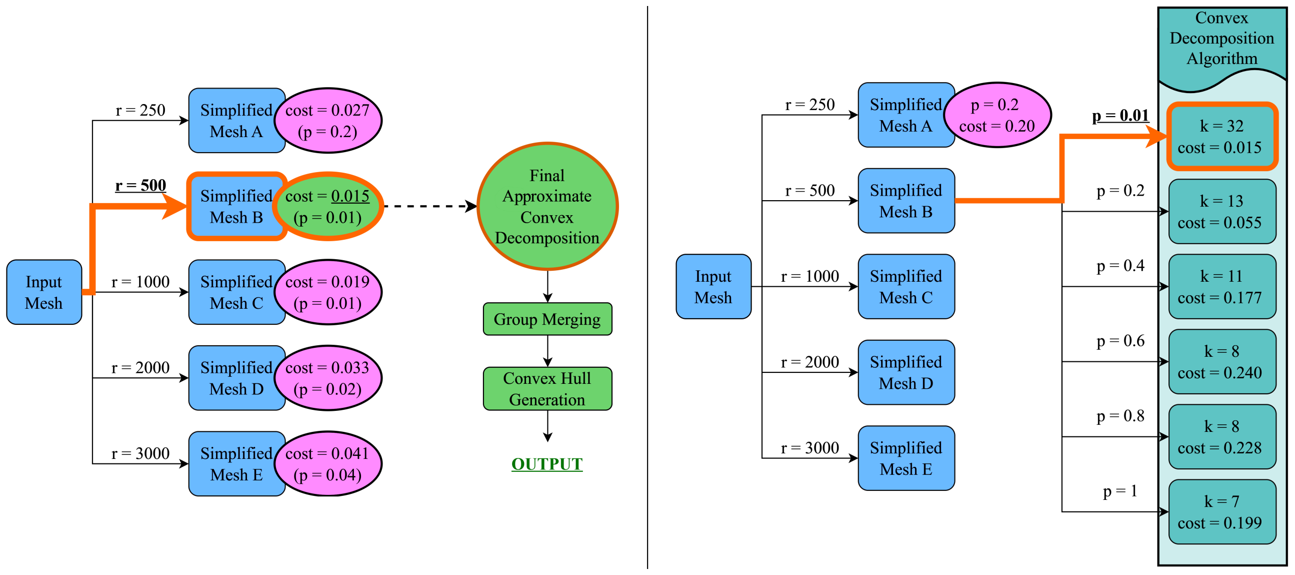

Figure 1. (Left) The first half of the method, threshold selection, with input parameters set to the values listed in subsection 4.3. The algorithm is represented in a tree-like structure, where the input mesh branches into the simplified meshes, each branching further using the possible planar thresholds. For each leaf node of the tree, the algorithm described in subsection 3.4.1 is executed. From the leaves of each simplification subtree that have a cost below the input concavity cost threshold (0.05 in this case), the leaf that has the smallest number of pieces in its decomposition (k) is selected as the candidate decomposition (pink) for that simplification level. The figure illustrates the execution of this step at the maximum resolution parameter $r = 500$. (Right) The second half of the method, resolution selection, follows the threshold selection step shown on the left side. After each simplification branch has selected a candidate, the one with the lowest cost is chosen as the output (green). Following the group merging step in subsection 3.6 and the generation of the convex hulls of the polygon groups in the result, we obtain our final output.

3.1. Input and Output

For the method described in this paper, the primary input required is a 3D triangle mesh that will undergo approximate convex decomposition. In addition to this, several input parameters are required to aid in selecting the final decomposition in subsections 3.4.2 and 3.5. These are the concavity cost threshold (ϵ) which controls the amount of concavity in the final result; the resolution cut-off parameter (μ) which is used to skip higher resolutions when the decompositions do not become more complex as the resolution increases; the simplification quality (Q) which affects the degree of simplification applied to the mesh; and a series of n maximum resolution parameters that are used to select the best option among several possible simplified versions of the input mesh:

$$R = \{r_1, r_2, r_3, ..., r_n\}. \tag{1}$$

The method results in a single output: the approximate convex decomposition of the input mesh. The convex hulls of each polygon group are represented by separate meshes, which, when combined, form the overall model.

Since our method works on the simplifications of a mesh, computation time primarily depends on the input maximum resolution parameters rather than the original resolution of the input mesh. It is also inversely proportional to the number of convex pieces found in the possible outputs. Fewer convex pieces imply that each piece has a greater polygon count, leading to an increased number of planes to compare with each neighboring polygon when deciding whether to add them to the convex group or not, during the formation of that piece.

Therefore, the time complexity is $O(N/k_{avg})$ where N is the sum of the maximum resolution parameters r_i, and k_{avg} is the average number of convex pieces across the decompositions achieved at each simplification level. However, due to the performance shortcuts mentioned in subsections 3.4.2 and 3.5, the computation time of the method is sometimes even faster than expected.

3.2. Mesh simplification

Finding an exact convex decomposition is known to be NP-hard and would yield too many convex pieces to be computationally beneficial. Approximate convex decomposition through mesh simplification leads to more desirable results. For a given iteration i of this method with the maximum resolution parameter r_i, we first simplify the mesh by the simplification quality Q (where $Q \in R$ and $0 < Q \leq 1$), and repeat this process until the polygon count of the mesh is less than or equal to r_i. In our experiments, mesh simplification was implemented using Unity Mesh Simplifier.[1]

3.3. Data structures

Once the mesh has been simplified, the next step would be to set up the data structures we use. The first of these is a list of “Triangle” objects. An instance of a triangle holds an ID, a normal, three vertex ID’s representing its vertices, the ID of the convex group it is in, and a boolean flag to discern whether the triangle in question is already within a polygon group or not.

The second data structure is a list of “Vertex” objects solely used to increase the algorithm’s time performance. This class holds a 3D coordinate designating the vertex’s position, and a list of adjacent triangle objects for easy access to the polygons that a particular vertex is part of.

To fill out these two lists, we first iterate through the vertices of the input mesh, adding each one as a new Vertex object into the vertex list. We then iterate through the polygons of the mesh and insert a new Triangle object into the triangle list for each polygon, calculating and storing the polygon’s normal in the object as well. During this iteration, each Vertex object related to a Triangle also has the corresponding Triangle object inserted into its adjacent triangle lists. The third data structure necessary for the algorithm is a list of integers we generate, that denote the adjacency data of each polygon. That is, for polygon i, its neighbors are the polygons with the ID’s given by the indices $3i$, $3i + 1$, and $3i + 2$ of this adjacency list.

3.4. The decomposition algorithm

In this section, we focus on the algorithm that takes a simplified input mesh, along with the necessary data structures to store information about it, and creates a grouping of its polygons to form convex pieces. This process is examined independently from the remainder of the method, which involves iteratively adjusting the mesh resolution and selecting the best result.

3.4.1. Convex polygon grouping

For a given planar threshold (p) and angular threshold (a), the mesh is processed in the main section of the algorithm that computes the convex polygon groups. The selection of these thresholds will be described in subsection 3.4.2.

[1]GitHub, Inc. (2024). Whinarn – Unity Mesh Simplifier [online]. Website https://github.com/Whinarn/UnityMeshSimplifier [accessed 06 July 2024].

We iterate over the Triangle objects stored in our triangle list using a queue structure. The first Triangle is inserted into the queue before entering the main loop. Within the loop, a triangle is removed from the queue. If it is not already in a convex group, it is added to the currently forming convex group, its group ID is stored within the triangle, and a flag is set to indicate that it is now in a group. A new Plane object (a built-in class found in Unity, our implementation environment, as mentioned in Section 4) is created to represent the plane formed by the triangle, which is then added to a list of planes.

Next, we iterate over the triangles w_0 to w_n that neighbor the current triangle v, where n represents the number of neighbors (acquired via the adjacency list data structure described in subsection 3.3). For each neighbor not yet in a group, we calculate the dot product of v_{normal} and a vector drawn from a vertex of v to the farther vertex of w, considering the 3D angle between the two triangles. If the dot product value is greater than the angular threshold a, w is found to be sufficiently concave with respect to the current convex group and is therefore disregarded, as seen in Figure 2a-left. Otherwise, w proceeds to the next check.

For each Plane object stored in the list of planes discussed earlier in this section (where each object holds the plane of a triangle within the current convex group), the perpendicular distance from the farther vertex of w to the plane is calculated. Taking the normal direction of the triangles as the positive side of the planes, if the vertex is located "behind" every plane, w is considered convex with respect to the rest of the group, as seen in Figure 2a-right, and the calculated distance for each plane will be negative. Conversely, if any of the distances to planes is greater than the planar threshold p, w is deemed sufficiently concave and is disregarded by the current convex group.

This loop continues until the queue is empty, indicating that the convex group can no longer add any neighboring polygons into it. At this point, the group is added to the decomposition list, and the next triangle in the triangle list that is not yet in a group is added to the queue, starting the building of the next convex group. This process continues until all triangles have been iterated over. The resulting list of convex polygon groups becomes the final outcome of this step, as seen in Figure 2b-left.

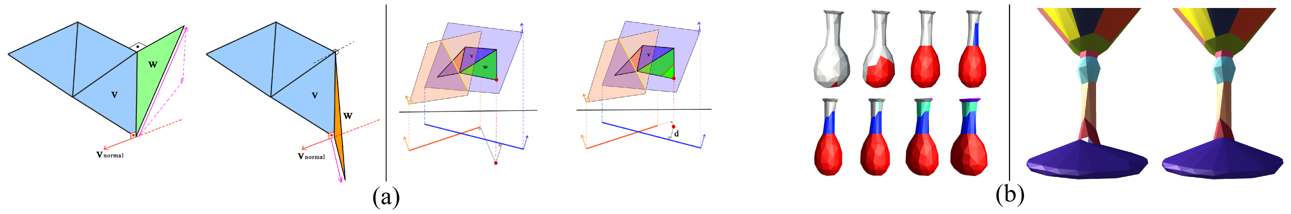

Figure 2. (a) (Left) Convex case where the neighbor w of the current triangle v is added to the current convex group, shown in blue. The dot product between v_{normal} and the vector from v to the farther vertex of w (pink) is nonpositive. (Right) Concave case where w is only added to the group if the dot product between v_{normal} and the pink vector is smaller than the angular threshold a. (Right) Cases where the neighboring triangle w (green) of the current triangle v (blue) is checked against the planes of the two other triangles in the convex group (orange and blue). The cross sections of the planes are also depicted, showing their distance to the farther vertex of w (red). The yellow line segment indicates the intersection of the two planes. (Left) In the convex case, the vertex lies behind each plane. (Right) In the concave case, the vertex lies behind the orange plane but in front of the blue plane. The intersection of w and the blue plane is shown in red, implying a bend outward; thus, w will not be added to the convex group unless the distance d is less than the planar threshold p. (b) Various steps of the polygon grouping process. When one group can no longer grow, the next group begins to form. (Right) Group merging removes small unnecessary pieces from the final decomposition. This is illustrated by the white piece generated in the left image, where the stem of the cup meets the base, which has been merged into the neighboring groups in the right image.

3.4.2. Threshold selection

The process described in subsection 3.4.1 is repeated with different p values, where

$$p \in \{0.01, 0.2, 0.4, 0.6, 0.8, 1\}. \tag{2}$$

The a threshold does not undergo this process and remains at a constant value of 0.01. Test results described in Section 4 indicate that, given the same set of possible values as p (Equation 2), 0.01 was selected for a in most instances, with negligible differences observed in the remaining cases. Among all the decompositions obtained through these values, we then calculate their concavity and select the most suitable candidate for the current resolution.

Given p_i for iteration i, a list of convex polygon groups G_i is created using the method described in subsection 3.4.1 as a possible decomposition of the mesh. In order to quantify the concavity of G_i, we utilize a cost function in the form of (Equation 3), as described in subsection 3.4.3. Tis function returns the cost of a single polygon group; therefore, we iterate over all the groups within G_i and calculate the cost of each one to determine its average concavity cost. During this process, groups containing two or fewer polygons are excluded, as they are likely to merge into larger neighboring groups in the merging step mentioned in subsection 3.6, and their contribution to overall concavity can be considered negligible due to their small size.

While iterating over the groups, an intermediate average cost is maintained by summing the costs calculated so far and dividing this total by the current number of iterations. This intermediate average is a performance measure used to eliminate the remaining cost calculations of iteration i if it seems likely that the resulting decomposition will be of high concavity, at which point we proceed to iteration $i+1$ using p_{i+1}. If at any point the intermediate average cost found on iteration i is greater than either three times the concavity cost threshold (ϵ), or greater than three times the minimum average cost found on iterations 0 to $i-1$, the cost calculation is aborted with the assumption that the cost has already risen too high to yield a viable candidate as output. Similarly, if it is determined that G_i consists of a single group, indicating that the decomposition using this p_i has failed, the remaining p values are not tested, and the loop for the current resolution is ended. Any future p values will imply a greater threshold and therefore will not further decompose the mesh.

Once the iteration of G_i is complete and its final average concavity cost is calculated, this value undergoes several checks to determine whether it will replace the existing candidate for the final output decomposition of the current resolution r_{curr}.

First, if the calculated value is found to be less than 10^{-6}, it is considered sufficiently convex to be immediately selected as the candidate decomposition for r_{curr}. Following this, a primary check is conducted, where this decomposition is stored as the candidate if and only if its average concavity cost is less than the concavity cost threshold ϵ and the number of convex groups it contains is fewer than that of the previously recorded candidate for the current resolution. In other words, among all the decompositions found for different values of p at a given resolution, the one that decomposed into the least pieces while still staying within the cost threshold ϵ is selected to be the outcome of said resolution. This approach aims to preserve the main aspects of the original mesh while achieving sufficiently simple convex pieces and avoiding overdecomposing.

There is a fail-safe in place in case no candidate decomposition can be selected by the end of this process for any reason. In such a case, we simply select the decomposition that resulted in the smallest average concavity cost.

3.4.3. Cost function

To calculate the convexity of a given surface, we could not use any volume-based techniques such as V-HACD [14] due to the fact that the convex polygon groups formed were almost always open meshes. For this reason, we have looked into the cost function used in HACD [13]. In this work, the concavity cost consists of two elements: a concavity factor that quantitatively measures a surface's concavity and a shape factor that promotes compact surfaces, where a surface is defined as a set of adjacent polygons. We have decided that the concavity factor used in HACD alone is sufficient as our cost function, making the cost of a given group of polygons:

$$cost(S) = \frac{argmax_{M \in S} \|M - P_S(M)\|}{D_S}, \tag{3}$$

where S is the set of vertices forming the given polygon group, or surface, D_S is the 3D bounding box diagonal of S for normalization, and $P_S(M)$ is acquired by projecting a point M to the convex hull of S, via the normal of M with respect to S. We compute convex hulls using the quickhull algorithm.[2]

3.5. Resolution selection

We mentioned in subsection 3.4.2 how the decomposition algorithm repeats itself using different planar thresholds, and how a final result out of these outputs is selected for a given resolution. This algorithm as a whole is then repeated for each of the r_i maximum resolution values given in the input set R (Equation 1). This leads to the same mesh input being simplified to varying levels, as described in subsection 3.2, and returning a decomposition candidate for each resolution. Our goal is to select one final output decomposition among these candidates.

One performance measure implemented is our input resolution cut-off parameter (μ). This value is a small positive integer, set to 5 in all experiments detailed in Section 4. After the candidate decomposition for a resolution r_i is found, if r_i is greater than 500 and the decomposition still only contains fewer than μ pieces, the remaining resolutions r_{i+1} to r_n are aborted. This decision is based on the assumption that the input mesh represents a simple shape that easily decomposes into only a few convex parts, rendering the computation of higher resolution decompositions unnecessary.

Following the computation of all the candidate decompositions, we aim to select the decomposition with the smallest average concavity cost. However, we encounter another challenge: the current method favors higher resolutions that may lead to overdecomposition of the mesh, as this often results in a lower cost. To address this issue, we calculate the average number of convex pieces contained in each candidate decomposition (k). Subsequently, all decomposition candidates consisting of greater than $k + \frac{k}{2}$ convex pieces are eliminated. This approach ensures that the candidate selection favors decompositions with a number of pieces closer to or below the average. The selected candidate then becomes the final decomposition.

3.6. Group merging

After obtaining the convex decomposition of the output, consisting of a set of adjacent polygon groups forming convex pieces, we perform a final step called group merging. In this step, any small convex group composed of one or two polygons is assessed for merging with neighboring groups. This is performed by checking if its polygons share two or three sides with a neighboring group. In this case, said polygons are made to be part of

[2]GitHub, Inc. (2024). OskarSigvardsson – unity-quickhull [online]. Website https://github.com/OskarSigvardsson/unity-quickhull [accessed 06 July 2024].

the larger neighboring group and removed from the original one. This is performed because such small groups of polygons are better merged into larger ones, to avoid the formation of tiny convex group islands that do not significantly affect the overall result. An example of this is shown in Figure 2b-right.

This merging process is repeated until no groups containing one or two polygons remain, or until such groups can no longer be merged with neighboring groups due to having three different groups on three different sides of its polygon(s). Additionally, a group consisting of one or two polygons will not be merged into its neighbors if the diagonal of its bounding box is sufficiently large to be considered significant. This prevents the incorrect merging of large surfaces composed of few polygons, which could result in deformations in the final mesh.

At this point, we have acquired the polygon groups that form our convex decomposition. We can then create meshes containing the convex hulls of each group, which together form the final output.

4. Experimental results

We have created a test dataset of meshes and conducted experiments to evaluate the performance of our method in terms of timing and accuracy. Additionally, we have implemented another published method for convex decomposition of 3D meshes and ran the same tests to enable a direct comparison, allowing us to identify the advantages and limitations of our algorithm.

As noted in Section 1, one application of convex decomposition is in game development, where it enables nonconvex objects to utilize rigidbody physics. Accordingly, the tests explained in this section were implemented within the Unity game development engine using the C# programming language.

4.1. Dataset

For the purposes of experimentation, we have compiled a test suite of meshes using Princeton University's 3D Mesh Segmentation Benchmark dataset [16]. This dataset comprises 400 meshes, divided into 20 categories, each containing 20 meshes. The meshes in each category have a similar theme, such as vases, planes, and animals. Each mesh was manually triangulated by 80 individuals to observe how humans decompose a model into functional pieces. For our dataset, we have selected 10 categories, with 10 meshes from each category, resulting in a total of 100 meshes. Additionally, a model homeomorphic to a sphere, named "Spot,"[3] was used as an experimental mesh.

4.2. Comparison algorithms

To compare our results with other published methods, we have used the public source of V-HACD [14] and implemented the HACD algorithm [13], from which we derived our cost function in subsection 3.4.3. The latter algorithm operates based on a graph data structure, where each node initially represents a surface composed of a single polygon from the mesh, and each edge denotes the adjacency between two polygons. For each edge, the cost of combining the surfaces at both of its nodes is then calculated using:

[3]Crane K (2024). 3D Model Repository [online]. Website https://www.cs.cmu.edu/~kmcrane/Projects/ModelRepository/#spot [accessed 06 July 2024].

$$
\begin{aligned}
cost(S) &= \frac{argmax_{M \in S} \|M - P_S(M)\|}{D_S} + \alpha E_{shape}(S), \\
\alpha &= \frac{\beta}{10 D_S}, \\
E_{shape}(S) &= \frac{\rho^2(S)}{4\pi\sigma(S)},
\end{aligned}
\tag{4}
$$

where S is the surface formed by the union of the surfaces within the two end nodes of an edge in the graph, D_S is the 3D bounding box diagonal of S, $P_S(M)$ is the projection of a point M to the convex hull of S, $E_{shape}(S)$ is a shape factor that facilitates more compact polygon grouping, α is a parameter that reduces the contribution of $E_{shape}(S)$ as the algorithm progresses, since the polygon groups begin to become concave, β is an input parameter that determines the concavity threshold, and $\rho(S)$ and $\sigma(S)$ represent the perimeter and area of S, respectively.

After this cost calculation, a half-edge collapse is performed on the edge that results in the smallest cost surface, combining both polygon surfaces into the remaining node and updating the costs of its surrounding edges. This process is repeated until no further edges that would result in a surface with a cost less than β remain. The resulting nodes each contain a surface that corresponds to a convex polygon group, effectively decomposing the mesh.

In our implementation, we modularly replaced the algorithm described in subsection 3.4 with this method, while maintaining the steps outlined in other sections, such as simplification, resolution selection, and group merging. This method was applied to ensure accurate comparisons by minimizing the number of variables, with only the actual calculation of the polygon groups changing in each iteration. Additionally, the resolution selection step was preserved because, similar to our method, different resolutions of meshes yield different results, and we aim to select the best one based on concavity cost and the number of pieces.

4.3. Results

In this section, we applied our method to the input meshes and presented the resulting outcomes. Each polygon group is displayed alongside its generated convex hull, with a unique randomized color assigned. All tests were conducted with the simplification quality (Q) set to 0.8, the concavity cost threshold (ϵ) at 0.05, the resolution cut-off parameter (μ) at 5, and the maximum resolution parameters defined as $R = \{250, 500, 1000, 2000, 3000\}$, resulting in a total of $N = 6750$. Unity was utilized to implement both our method and the comparison algorithm, as well as to visualize the results.

In addition to the figures illustrating our outcomes, we also present Table that details relevant data regarding the decomposition process. The data provided includes the following: the number of convex pieces found, the average number of pieces found across all resolutions, the computation time, the average concavity cost, the selected planar threshold p, the maximum resolution r, the final simplified polygon count, and the initial polygon count of the mesh.

From the results presented in Figure 3-left, it is evident that our algorithm successfully distinguishes convex features, including the limbs of a human, the wings of a plane, the legs of a chair, and the humps of a camel. In curved structures, such as the cup or the curved tentacles of an octopus, the mesh is divided into multiple small convex pieces to account for continuous concavity. Moreover, the shapes all retain their original basic structure. Notably, small gaps, such as those between the arms and torso of the human figure or between the fingers on the hand, are properly maintained, and the hollow structure of the cup is also mostly preserved.

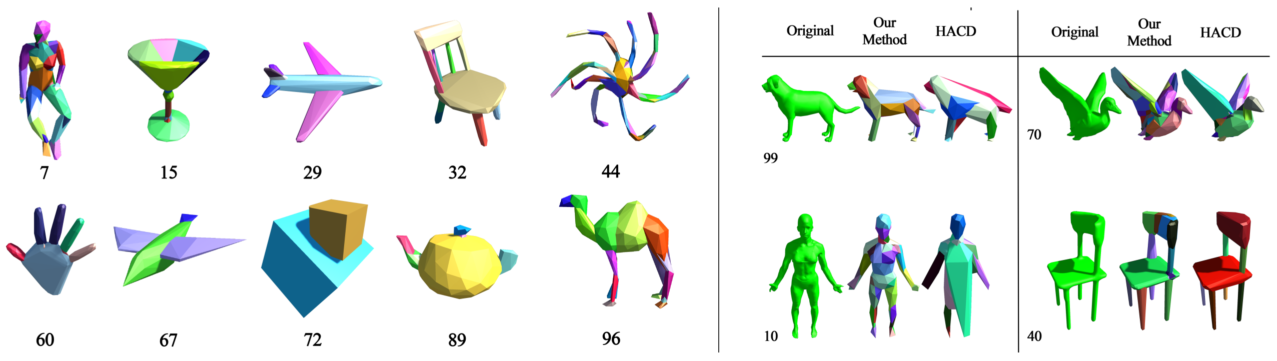

Figure 3. (Left) An example result from each model category in our input database, with models labeled by their dataset numbering. (Right) Comparative outcomes between our method (center) and the results generated by the comparison algorithm HACD (right) for the same input mesh, with the original mesh shown for reference (left). Models are labeled according to their dataset numbering.

Table 1. *
Data related to the results shown in Figure 3-left.

Model	Number of pieces	Average number of pieces	Computation time (s)	Cost	p	r	Polygon count	Initial polygon count
7	38	58	11.27	0.009591	0.01	500	494	11258
15	15	39	8.16	0.021415	0.2	500	430	30008
29	12	19	13.05	0.012669	0.6	500	462	13170
32	18	43	9.67	0.011501	0.8	500	450	31456
44	53	68	5.89	0.009776	0.01	500	442	12646
60	20	44	14.29	0.016998	0.6	1000	990	3026
67	6	5	4.75	0.025047	0.4	250	212	11936
72	2	2	0.88	0.000007	0.2	250	218	29994
89	8	31	43.25	0.007944	0.2	250	246	13842
96	21	49	6	0.035519	0.2	500	436	19510

Examining Table, we observe that cases where 0.01 is selected as the planar threshold corresponds to models with small gaps, specifically the human and the octopus. This occurs because these gaps are quickly filled when higher thresholds are applied.

The concavity costs are consistently low, particularly for model number 72, where the simple nature of the shape appears to facilitate efficient decomposition. In this specific test, the resolution cut-off parameter (μ) was used to skip resolutions from 1000 to 3000, as the number of pieces remained less than μ=5 after the second iteration with r=500, resulting in the low time required.

4.3.1. Comparisons

As discussed in subsection 4.2, we have selected HACD [13] as the established algorithm for comparison with our method as an evaluation metric. For this purpose, all the input meshes were processed using HACD, and some of these results, along with the corresponding decomposition generated by our method, are displayed in Figure 3-right.

From the comparisons presented in the right column, it is evident that HACD attempts to group vertices in a more simplified way. For instance, in the chair model, the entire back is represented as a single piece, while our method decomposes the curved surface of the chair's back into multiple pieces. The same logic applies to the wings of the bird and the neck of the vase.

However, the approach employed by HACD is not always advantageous compared to our method, as illustrated by the examples in the left column. While attempting to simplify the convex pieces, HACD has resulted in significant deformation, whereas our model maintains the overall shape of the input meshes. In particular, HACD has filled the cup with a solid hull. This behavior is also evident in other similar examples; while our method effectively decomposes hollow objects, HACD tends to fill them in, as observed in Figure 4-left.

We highlight the main advantages of our method over HACD, specifically the lower concavity cost and reduced time required for each case. In particular, the latter is significantly better in certain cases, such as models 82 and 99, where our method achieved the convex decomposition much more quickly than HACD. Conversely, HACD produced decompositions with smaller resolutions and fewer pieces than our method in some cases, but not in all. This observation is consistent with the earlier results discussed.

In addition to HACD, we utilized V-HACD [14] as an alternative comparison algorithm, employing its released library. The results can be seen in Figure 4-right.

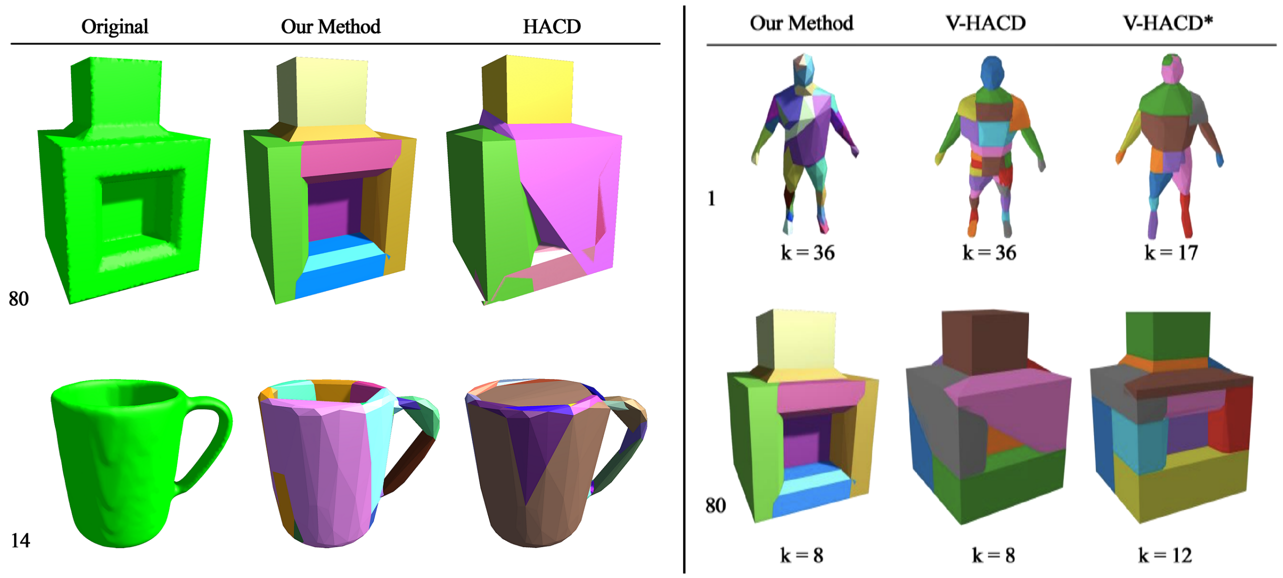

Figure 4. (Left) Comparison of results for hollow input meshes, showing outputs from our method (center) and those generated by HACD (right), with the original mesh for reference (left). Models are labeled according to their dataset numbering. Our method mostly or fully preserves hollow spaces (e.g., the interiors of cups), while HACD tends to fill these spaces. (Right) A comparison using a different method, V-HACD. One key input for V-HACD is the target number of convex hulls the algorithm aims to generate in the final output. We first decomposed each input mesh into the same number of pieces (k) as in our method (center). In the column marked with an asterisk (*), we adjusted this parameter until significant features—such as the hands in model 1 or the hole in model 80—became visible. For model 1, V-HACD required fewer pieces than our method to achieve a satisfactory decomposition. However, for model 80, neither the initial nor the adjusted number of pieces allowed V-HACD to produce a better decomposition than our method. Thus, while V-HACD can yield more accurate results in certain cases with trial and error, it may be less effective in others due to the lack of automatic parameter selection.

4.3.2. Ablation study

Finally, we will compare our method to itself by keeping all variables constant except for one and observing the results. Specifically, we aim to examine the differences arising from changing the two main factors of our method: the planar threshold and the maximum resolution parameter.

First, we will fix the maximum resolution to 500 and generate decompositions using every planar threshold employed in our algorithm (Equation 2). Selecting a few example inputs, the results are presented in Figure5-right.

In the usual operation of our algorithm, we would select the decomposition with a cost lower than the specified concavity cost threshold, ϵ, while minimizing the number of pieces to identify the candidate for this maximum resolution. If no suitable decomposition is found, we would then choose the result with the lowest cost.

Data on the decompositions using each planar threshold for Model 2 reveals that at $\epsilon = 0.05$, the value used in our tests, we would select the first option with $p = 0.01$ as it is the only option below the cost threshold. However, if ϵ were set to be 0.1, the decomposition with $p = 0.2$ would be chosen due to its fewer pieces. The reason the timing increases as p increases is that with larger and less decomposed convex groups, there are more polygons to consider during the planar calculations when adding a new polygon. We will then fix the planar threshold at 0.1 and apply additional input models through the algorithm for each maximum resolution parameter used in our method.

In the usual operation of our algorithm, this step involves finding the average number of convex pieces k, removing decompositions with more than $k + \frac{k}{2}$ pieces, and selecting the one with the lowest average concavity cost among the remaining decompositions as the final result.

5. Limitations

The success rate of our method is highly dependent on the quality of the input parameters, particularly the maximum resolution settings that determine which simplified versions of the input mesh will be tested to identify a satisfactory output. If the specified resolutions are too low, the mesh may become overly deformed, leading to an unviable output. Conversely, if the resolutions are excessively high, our method may yield an excessive number of unnecessary pieces, complicating the decomposition process. This reasoning extends to other input parameters as well.

Examining our experiments and comparisons in Section 4, we observe cases where the decomposition exhibits noticeable convex features that have been separated; however, some pieces still appear to be combinable into simpler forms. Thus, while the resulting decompositions are adequate, there is potential for improvement by eliminating unnecessary pieces in certain cases, as demonstrated by our comparisons to HACD. This behavior is particularly evident when no simplification is applied to the input mesh, as illustrated in Figure 5-left.

Our method is likely to produce overlapping convex groups due to the use of convex hulls for polygon groups. While this may not pose a problem in certain applications, such as collision detection where collisions between these convex hulls can be disregarded, it could be undesirable in other contexts, such as in 3D printing [24].

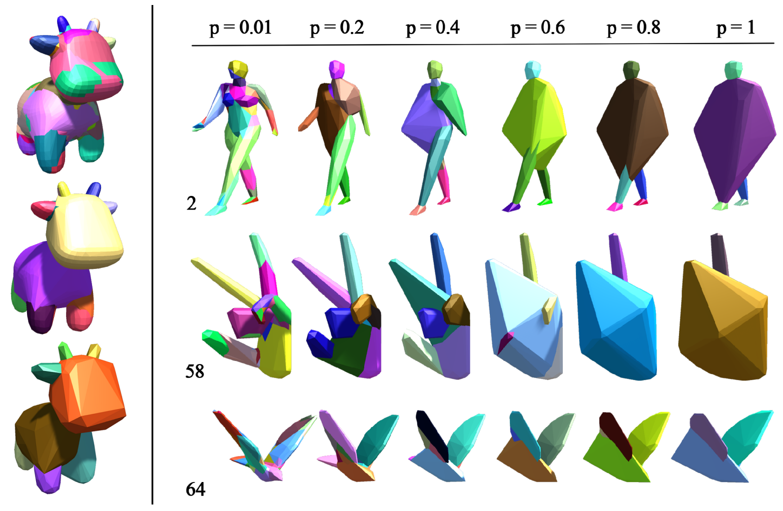

Figure 5. (Left) Failure case using the input model "Spot." In this experiment, no mesh simplification or resolution selection was performed. Our method (top) resulted in a more overdecomposed model compared to the result of HACD (center). However, our method completed this decomposition in approximately 1 min, while HACD required over 14.5 h. This issue was resolved by applying simplification with r = 900, enabling our method to generate a significantly improved decomposition (bottom) in just 1.75 s. (Right) Evolution of several input models at a fixed maximum resolution ($r = 500$), as the planar threshold (p) is varied through all possible values. Models are labeled according to their dataset numbering.

6. Conclusion and future work

We have presented a method for decomposing 3D meshes into convex parts by iterating over the polygons of the mesh from a bottom-up perspective. Through multiple iterations with varying thresholds, we select the most effective decomposition to ensure an output that balances the formation of simple convex pieces with the preservation of the original mesh shape. Although some input meshes may result in existing methods outperforming ours in terms of accuracy, a significant advantage of our approach is the reduced time required to achieve a sufficiently good convex decomposition.

Future work may involve adjusting the core components of our method, including the selection of maximum resolution parameters, the planar thresholds for generating candidate decompositions, and the threshold values for simplification quality and cost concavity. Adjusting these parameters could lead to higher-quality decompositions for certain input meshes, making it beneficial to conduct additional tests to identify the optimal combination of values for our algorithm. Moreover, other promising directions for future research include computing convex decomposition in dynamic scenes and exploring 3D data representations beyond triangle meshes, such as quad meshes, NURBS, and point clouds. Each maximum resolution parameter corresponds to an independent iteration of our algorithm, suggesting that parallel programming can be effectively integrated into our approach.

Acknowledgment

This work was supported by The Scientific and Technological Research Council of Türkiye (TÜBİTAK) under project EEEAG-119E572.

References

[1] Liu G, Xi Z, Lien J-M. Nearly convex segmentation of polyhedra through convex ridge separation. Computer-Aided Design 2016; 78: 137-146. https://doi.org/10.1016/j.cad.2016.05.015

[2] Attene M, Mortara M, Spagnuolo M, Falcidieno B. Hierarchical convex approximation of 3D shapes for fast region selection. Computer Graphics Forum 2008; 27 (5): 1323-1332. https://doi.org/10.1111/j.1467-8659.2008.01271.x

[3] Deng B, Genova K, Yazdani S, Bouaziz S, Hinton G et al. CvxNet: learnable convex decomposition. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); Seattle, WA, USA; 2020. pp. 31-44, https://doi.org/10.1109/CVPR42600.2020.00011

[4] Chen Z, Tagliasacchi A, Zhang H. Bsp-net: Generating compact meshes via binary space partitioning. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); Seattle, WA, USA; 2020. pp. 45-54, https://doi.org/10.1109/CVPR42600.2020.00012

[5] Thul D, Ladický L, Jeong S, Pollefeys M. Approximate convex decomposition and transfer for animated meshes. ACM Transactions on Graphics 2018; 37 (6): 1-10. https://doi.org/10.1145/3272127.3275029

[6] Lien J-M, Amato NM. Approximate convex decomposition of polygons.In: Proceedings of the Twentieth Annual Symposium on Computational Geometry; Brooklyn, NY, USA; 2004. pp. 17-26. https://doi.org/10.1145/997817.997823

[7] Lien JM, Amato NM. Approximate convex decomposition of polyhedra. In: Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling; Beijing, China; 2007. pp. 121-131. https://doi.org/10.1145/1236246.1236265

[8] Ghosh M, Amato NM, Lu Y, Lien J-M. Fast approximate convex decomposition using relative concavity. Computer-Aided Design 2013; 45 (2): 494-504. https://doi.org/10.1016/j.cad.2012.10.032

[9] Liu R, Zhang H, Busby J. Convex hull covering of polygonal scenes for accurate collision detection in games. Graphics Interface 2008; 203-210.

[10] Yang X, Jia X. Simple primitive recognition via hierarchical face clustering. Computational Visual Media 2020; 6: 431-443. https://doi.org/10.1007/s41095-020-0192-6

[11] Saha S, Biswas A. Approximate convex decomposition of 3D digital object surface using scalar triple product of vectors. In: The International Symposium on Artificial Intelligence and Mathematics (ISAIM); Fort Lauderdale, FL, USA; 2022.

[12] Li Z, Zhang Z, Liu H, Yang L. A new path planning method based on concave polygon convex decomposition and artificial bee colony algorithm. International Journal of Advanced Robotic Systems 2020; 17 (1). https://doi.org/10.1177/1729881419894787

[13] Mamou K, Ghorbel F. A simple and efficient approach for 3D mesh approximate convex decomposition. In: 16th IEEE International Conference on Image Processing (ICIP); Cairo, Egypt; 2009. pp. 3501-3504. https://doi.org/10.1109/ICIP.2009.5414068

[14] Mamou K. Volumetric hierarchical approximate convex decomposition. In: Lengyel E (editor). Game Engine Gems 3. Boca Raton, FL, USA: CRC Press, 2016, pp. 141-158.

[15] Wei X, Liu M, Ling Z, Su H. Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search. ACM Transactions on Graphics 2022; 41 (4): 1-18. https://doi.org/10.1145/3528223.3530103

[16] Chen X, Golovinskiy A, Funkhouser T. A benchmark for 3D mesh segmentation. ACM Transactions on Graphics 2009; 28(3): 1-12. https://doi.org/10.1145/1531326.1531379

[17] Ghosh M, Thomas S, Amato NM. Fast collision detection for motion planning using shape primitive skeletons. In: Morales M, Tapia L, Sánchez-Ante G, Hutchinson S (editors). Algorithmic Foundations of Robotics XIII. WAFR 2018. Springer Proceedings in Advanced Robotics, volume 14. Cham, Switzerland: Springer, 2020, pp. 36-51. https://doi.org/10.1007/978-3-030-44051-0_3

[18] Taştan O, Sahillioğlu Y. Human body reconstruction from limited number of points. Computer Animation and Virtual Worlds 2021; 32 (5): e1995. https://doi.org/10.1002/cav.1995

[19] Sahillioğlu Y. Recent advances in shape correspondence. The Visual Computer 2020; 36 (8): 1705-1721. https://doi.org/10.1007/s00371-019-01760-0

[20] Asiler M, Sahillioğlu Y. KerGen: a kernel computation algorithm for 3D polygon meshes. Computer Graphics Forum 2024; 43 (5): e15137. https://doi.org/10.1111/cgf.15137

[21] Dai H, Amice A, Werner P, Zhang A, Tedrake R. Certified polyhedral decompositions of collision-free configuration space. The International Journal of Robotics Research 2023; 43 (9). https://doi.org/10.1177/02783649231201437

[22] Vavilala V, Forsyth D. Convex decomposition of indoor scenes. In: IEEE/CVF International Conference on Computer Vision (ICCV); Paris, France; 2023. pp. 9142-9152. https://doi.org/10.1109/ICCV51070.2023.00842

[23] Asiler M, Sahillioğlu Y. 3D geometric kernel computation in polygon mesh structures. Computers & Graphics 2024; 122: 103951. https://doi.org/10.1016/j.cag.2024.103951

[24] Mert LM, Yaman U, Sahillioğlu Y. A fabrication-oriented remeshing method for auxetic pattern extraction. Turkish Journal of Electrical Engineering and Computer Sciences 2020; 28 (3): 1535-1548. https://doi.org/10.3906/elk-1908-51