Semantic Overlap Free Layouts of Surface Parameterizations

Josh Kalapos
Advised by Rohan Sawhney and Professor Keenan Crane

May 12th, 2019

Abstract

Computing UV parameterization is a fundamental task in geometry processing with applications in texturing, machine learning and 3D fabrication. In general, the goal is to take a surface and map it to a plane while minimizing the distortion. A recent approach that drastically reduces distortion in parameterization uses the method of cone singularities. Rather than mapping the surface directly to a plane, we can find an intermediate polyhedron that better approximates the surface. Afterwards, the polyhedron can be laid out in the plane by cutting through a spanning tree that passes through the taken cones. However, this pipeline does not guarantee a layout that does not overlap itself. Tackling this problem of layout does not need to take distortion into account, since the shapes have already been flattened. We explore ways to partition the layout so that it is free of overlaps and the separated components are few and semantically reasonable.

1 Introduction

1.1 The Problem

A conformal surface parameterization maps a curved surface to the plane while preserving angles [Cra18]. These kind of maps have been historically important in cartography, and in present day are widely used for texturing, machine learning, and other geometry processing tasks. When conformally flattening a surface, though angles are preserved, areas on the surface may become distorted. The goal then, is to flatten a surface with the least amount of area distortion possible. One method to reduce distortion while flattening a surface is to choose cone singularities and allow for cuts on the mesh through these cones [SSC18]. Intuitively, physically trying to push or unwrap a surface down to the plane becomes significantly easier when the surface can be torn or cut. Technically, using cone singularities allows for a mesh to be mapped onto an intermediate polyhedron. Any convex polyhedron can be unwrapped onto the plane [OF], without any additional stretching of faces, or any conformal distortion. The flattening provides an intrinsic triangulation, meaning that there is no information about how the polyhedron is embedded in space but there is information about the edge lengths of the triangulation. Although these type of flattenings are quick to compute and with recent methods can allow for control of the parameterization's boundary, they do not guarantee that overlaps do not exist. Meaning that when the mesh is taken to the plane, two or more points on the three-dimensional surface are mapped to the same location. Such an outcome is not desirable because, for example, multiple parts of a texture map can access the same spot of an image. Another case is when fabricating 3D objects from flat sheets of material, there needs to be a one to one correspondence between the sheets and the 3D object. Producing a parameterization that is free of overlaps is a necessity in nearly all use cases.

A solution then, would be to prevent overlaps from being created in the flattening algorithm. This however, requires non-linear optimizations, which are much slower than any of

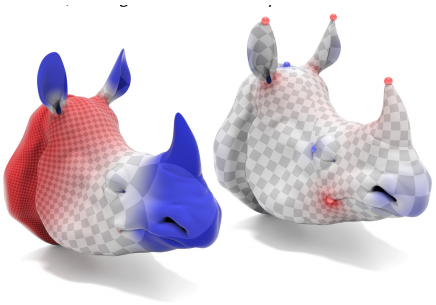

Figure 1: Left: Without cone singularities, conformal flattenings can have extreme area distortion, highlighted by blue and red areas. Right: By placing a few optimally chosen cone singularities on the surface, distortion can be drastically reduced. [Figure taken from Optimal Cone Singularities for Conformal Flattening]

Figure 2: Paramaterizing with cones is the same as taking a smooth surface and flattening it onto a polyhedron. Then, this polyhedron can be unwrapped onto the plane with no additional distortion. [Figure taken from Optimal Cone Singularities for Conformal Flattening]

the linear flattening methods offered. Even more so, a method introduced by Sawhney and Crane [SC17], Boundary First Flattening (BFF), is a linear method for conformal parameterization that performs better than the other linear methods, while also allowing for control of the shape of the boundary. As a result, it becomes worthwhile exploring techniques to take the flattening provided by BFF that may have overlaps, and attempt to remove them.

1.2 Related Work

When developing an approach to eliminate paramaterization overlaps, what emerged as one of the largest considerations is what can be considered a "good" flattening. Creating a flattening that a subjective measure of "goodness" was difficult, as for a good amount of

time the direction of the approach was pretty unclear. Initially, some sources proposed the creation of paramaterizations that were extremely space efficient. One method, called Box Cutter [LVS18], focused on taking in a flattening, and optimizing the paramaterization to be as densely packed into a rectangle as possible. The method had some advantages, one of which was that as an initial preprocess, any overlaps were removed by turning them into distinct components and relocating them. A disadvantage was that after identifying any overlaps, which can be performed quickly, Box Cutter employed a relatively expensive graph algorithm that would take away from the practical benefits of BFF's efficiency. Furthermore, the packing that occurred afterwards seemed to breach what would be viewed as a "good" flattening, primarily because the largest priority was set on packing in the densest way possible. Especially in the case when the initial flattening has an overlap, the result observed seemed to have lost any visual meaning.. For example, an overlapping flattening of a body would be chopped up and would lose any distinction between limbs and the body. In some use cases, it is true that a flattening does not have to make visual sense. When baking lighting effects into a texture map, the paramaterization only serves as a way to store information without much human interaction. In this way, the direction of flattening becomes sort of simple, as very measurable aspects such as distortion, boundary length, packing efficiency can be used to grade a paramaterizations quality.

On the other hand, flattenings can be graded based on how they compare to how a human would like to see a mesh flattened. This approach is awkward to evaluate, as humans are somewhat picky and there exists an incredible variety of objects that can be flattened. A similar issue was explored with mesh segmentation with the Princeton Benchmark [CGF09]. Mesh segmentation is the process of decomposing a mesh into multiple smaller sub-meshes. Most often, these smaller pieces are meaningful, such as dividing up a mesh of a chair into its main frame and legs. With the introduction of the Princeton Benchmark, Chen et al. experiment with many different segmentation algorithms on a large set of meshes, and score them against segmentations done by hand. They conclude that people are surprisingly in

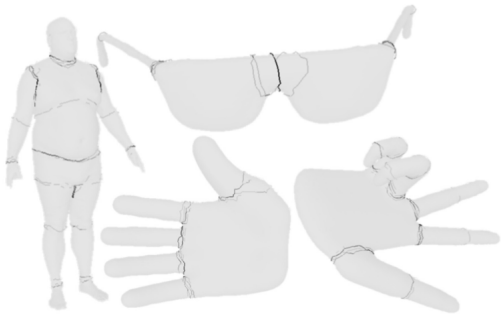

Figure 3: Meshes here are segmented by humans. Darker lines indicate agreement between multiple people. [Figure taken from A Benchmark for 3D Mesh Segmentation]

agreement with each other when it comes to coming up with segmentations (Figure 3), but that no one automatic segmentation algorithm is better than others for all types of objects.

It should not be too difficult to imagine how measuring the human aspect of mesh segmentation could relate to measuring the human aspect of mesh flattening. Taking the observations from the Princeton Benchmark, a reasonable approach to removing overlaps would be to first segment the mesh into meaningful pieces, and then to separate the flattening into these pieces. This does not necessarily directly remove overlaps, but it comes from a human observation that these overlaps occur often along boundaries where a mesh could be divided into sub-components.

1.3 Approach

Perhaps then, it could be worthwhile exploring how well this method could work. Most segmentation algorithms work relatively fast in practice, so adding segmentation to the back of the BFF pipeline would not introduce large computation costs that wash out the benefits of BFF's speed. So then, the general approach becomes pretty simple to describe. When given a mesh, BFF will produce some 2D paramaterization. If that parameterization does not contain any face overlaps, then everything is left as is. If there does exist overlaps in the flattening, such as in Figure 4, then the mesh gets segmented into meaningful sub-meshes. These smaller components then are separately laid out face by face and then packed into a

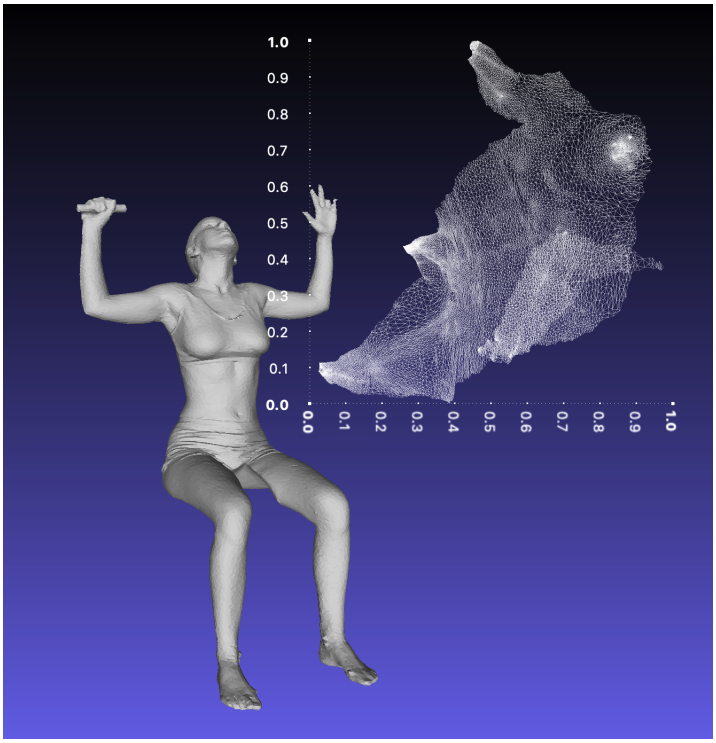

Figure 4: A lady flattened into the plane. Some distortion can be viewed by suboptimal cone placement and, more importantly, some overlaps between the flattened arm and body

uv map.

2 Semantic Layouts

The approach provided does not require any specific segmentation algorithm. Ideally, any kind of segmentation strategy could be switched in and, as long as it provides a labeling of which face belongs to which component, the process would remain the same. Before any segmentation occurs, BFF and some preprocessing has to occur to deem if any segmentation is necessary. By running Boundary First Flattening on a mesh, each vertex on the mesh gets mapping to uv coordinate(s) on the plane. Vertices, even though they have a single position in three-dimensional space, can have multiple positions on the plane depending on if they lie on the boundary or on segmentation cuts. This uv information is first used to determine if there are any overlaps in the parameterization. Finding out if any face overlaps another face

can be accomplished quickly by first inserting faces into a regular grid, and then checking if faces intersect with faces in their region. This helps reduce the number of intersection tests that need to be computed. If there are any overlaps detected, the process elects to segment the mesh in an attempt to eliminate them.

2.1 Segmentation

The segmentation strategy that gets implemented here is a mesh variation of k-means. In an overview, k faces get chosen to be a center of a segment, and through a region growing process, there comes to be k continuous components on the mesh. New centers get chosen and the region growing process is done again. Doing this process repeatedly until convergence or a iteration limits provides the final set of k segments. To have a semantic segmentation however, it is important to consider how to choose potential centers and how to grow regions. For the initial centers, there actually is not a great difference in the results between random initial centers and intelligently picked centers. Though the results are similar, well picked centers will make the algorithm converge faster, so they are still worthwhile calculating in order to save some computation time. A suggested method [STK02] is to use a furthest point algorithm. On the mesh, any face can be chosen as the first center. Then, the furthest face can be found by running Dijkstra's algorithm from the initial face. The last face to be explored is the furthest face. In this case, the distance between two adjacent faces is calculated by taking their centroids and taking the euclidean distance between them. Calculating their exact distance along the surface is a bit expensive, though a different way to calculate distance would be to take the distance from the centroid of one face, the center of the edge shared by the two faces, and the centroid of the other face. This would be a more accurate estimate for surfaces that are very wrinkly, but these centers do not exactly have to be perfect so this method was not added. The last face is added as another center. The process is repeated again except that there are now multiple sources in the search. More centers are then found until there are k centers chosen.

Now that there are k initial seeds for the segmentation, an iterative region growing process can proceed until it reaches convergence or it reaches some iteration limit. These processes in the discrete setting are not guaranteed to converge [JKS05], so some iteration limit should be set. The region growing process that follows is not too different from picking the centers, except that the distance function is changed and the faces get labeled as Dijkstra's Algorithm searches outward. For distance, in addition to calculating the straight-line distance between adjacent faces, an additional consideration is added for the difference in the normals of the faces. After calculating the unit normal for each face n_1, n_2, their agreement can be calculated by $1 - |n_1 \cdot n_2|$. The dot product should be one when the normals are pointing in the same direction and zero if the normals are perpendicular. The absolute value protects against when the normals are pointing in different directions as two faces with normals pointing in opposite direction are still parallel. This angle distance can be added to the original distance with some experimentally found weight. Having a consideration for changing angles draws out segment boundaries along concave seams, which is often where human segments occur. With the new distance function, Dijkstra's can once again be run with multiple sources, and faces are assigned to their closest center.

Before the next iteration, new centers have to be computed. With each iteration, these centers should be ideally be drawn towards some balance on the surface, which will produce the final segmentation. One technique to define a new center of a component is to find the face furthest from a components boundary. By adding all the faces as sources for Dijkstra and searching only inward, the final face visited will be the new center. Although this technique seems to provide a good notion of the new seed, a simpler strategy often produces better results in practice. By taking the average position of all the centroids of a component, a new center can be chosen by taking the face closest to the average position. This method is faster and, surprisingly, some research says it produces better segmentation results [STK02]. After choosing new centers, region growing is carried out before once again choosing new centers. To check for convergence, the number of faces that have changed to a different component

between iterations is divided by the total number of faces. If this value is below five percent, then terminate. After either converging or reaching to iteration limit, every face now has a label that indicates which sub-mesh it belongs to.

2.2 Creating a New Paramaterization

After producing a segmentation, each sub-mesh can be laid out separately on the plane. Though to this post-process what BFF produced was the uv coordinates of vertices, the important information stored was the edge lengths of each face. When the components get placed again, their uv coordinates will change, but each face's edge length will remain the same. Taking in multiple mesh components and relaying them out can lead to multiple different approaches. One would be to grow face by face, checking if a newly laid face overlaps with the component. This method introduces some difficulties like creating many small uv islands created when a few faces cannot find a valid fit. Instead, a placement strategy that does not repeatedly check for overlaps is implemented. Though this method does not guarantee total overlap removal it does run quickly and serves its purpose until the previously mentioned technique can be refined. The exact process is once again another form of region growing. An unvisited face is first found and laid onto the plane. Then, its neighbors are added to a queue. Faces connected to their neighbor using edge lengths and their unvisited neighbors also get added. Growing stops when an edge is a boundary of the surface or if it is a cut between two different components. Only connectivity matters here so Breadth First Search is fine. Now that each face has a new uv position, these components can moved around with some bin packing algorithm to add some additional space efficiency.

3 Evaluation

This approach was evaluated against multiple different meshes that are provided by the Princeton Benchmark website. Though this benchmark does provide some tools to compare a

segmentation algorithm's effectiveness compared to human segmentations, the segmentations produced here were judged mostly visually. In this use case, where segmentation is an intermediate step to produce a better flattening, it somewhat made sense to use this level of judgement for now. Additionally, as said before, most humans tend to agree on cut locations so an individual's judgement is not a bad estimate on a segmentation quality.

3.1 Setup

To visualize the segmentation results, the Boundary First Flattening viewer was modified so each separate component was colored differently after segmentation. This viewer made it easy to load in and test on multiple different meshes. To view the new layout, the output needs to be viewed in a separate software that can display uv maps, such as MeshLab. The reason behind this is that the BFF viewer does not provision for single meshes being broken up into multiple components. As a result, any component packing occurs when the output is written to a file. So this file can be opened by some external program. Evaluation can now be done at the two different stages of the approach, the segmentation and the layout.

3.2 Results

For the segmentations, a lot of testing was done on meshes of living creatures, which may introduce some bias towards better segmentations of these studied models, but not of other objects such as chairs. Some runs were done on models of household objects, but the more interesting results were with segmentations of people. Humans can be separated into their limbs, torso and head, and this remains consistent with just about any model of a full body. As seen in Figure 5, the segmentation approach seems to follow such intuitions.

After getting layed out in 2D space once again, final results often took a similar appearance to Figure 6. By just quickly segmenting the mesh and laying out the faces, a great deal of cases that have overlaps can be eliminated. Some pieces currently look smaller than they should be, which can be fixed with better cone placement. Each piece laid out has some

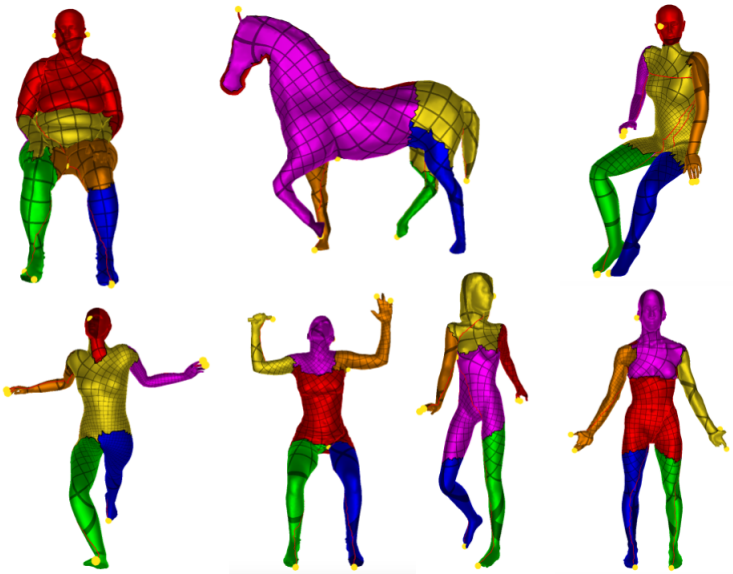

Figure 5: Example results of segmented meshes. Models are the same used by the Princeton Benchmark

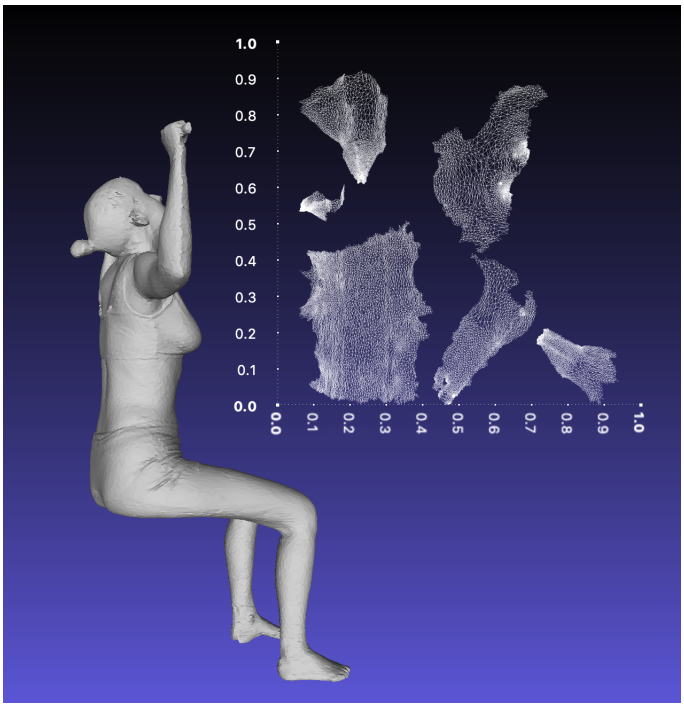

Figure 6: Six separated components take from the same flattening as Figure 4.

visual significance that was determined by the segmentation algorithm.

4 Observations

What becomes clearly apparent that for meshes of people, segmentations varied when a person was not situated in a standard position. Meaning a person whose one leg was bent would have a slightly different set of segment boundaries, primarily because the angle distance to traverse is larger. Some possibilities to fixing this issue could be taking some more costs into account like the growing gaussian curvature of the component, or potentially some more global strategy for segmentation done in other research could be used. Additionally, boundaries between segments are a bit jagged, which may mess with some applications. A future solution is to take endpoints of boundaries are straighten them out with shortest paths. Or, in the case that the boundary is a loop, using minimum cost cuts around the sruface [JKS05]. Perhaps the most important observation is that this method still doesn't eliminate all overlaps. Perhaps there is a way to guarantee there are no overlaps while also not creating any small islands and having efficient performance. There is still a lot of potential for creating smart algorithms that move around triangles to create a much better layout. Having a flattening as an intrinsic triangulation opens up many possibilities for games to play.

5 Surprises

Some surprises in this project include the difficulty of getting some consistent segmentation results over many meshes. There are many segmentation algorithms out there, and they always seem to hand pick test cases that show off where their algorithm succeeds. Additionally, many of these segmentation algorithms do not exactly fulfill the same purpose as what is needed in this paper. For example, some segmentations produce developable patches, meaning segments that are as flat on the surface as possible. With meshes that are very curvy,

the number of these patches required explodes and is not ideal for laying out as components. Knowing the correct number of components became a user problem, as it was simpler and far more effective for a person to directly specify the number of pieces a mesh should be split into. People usually make sense to have six pieces. Octopuses can have nine. Walking through many subjective problems meant for a lot of time spent testing out some technique, before finding out that it does not produce a visually satisfying output.

6 Conclusions

Although this explored method of overlap removal has experienced some success, there is still a lot of improvements to be made on its reliability. An improvement would be to ensure that the final result does not contain paramaterization overlaps, in addition to the semantic pieces provided. In certain trials, detecting overlaps dynamically as patches were built was expensive, but there is potential for accelerating the process, or constructing some other construction algorithm that requires less overlap queries. Much of the efforts here can be extended to fixing parameterizations not provided by just BFF. Given any flattening, and access to the original mesh, a semantic layout could be created. This ability could prove to be useful if another uv paramaterization algorithm needs overlap removal as a post process.

References

[STK02] Shymon Shlafman, Ayellet Tal, and Sagi Katz. "Metamorphosis of Polyhedral Surfaces using Decomposition". In: *Computer Graphics Forum*. 2002, pp. 219–228.

[JKS05] Dan Julius, Vladislav Kraevoy, and Alla Sheffer. "D-charts: Quasi-developable mesh segmentation". In: *In Computer Graphics Forum, Proc. of Eurographics*. 2005, pp. 581–590.

[CGF09] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. "A Benchmark for 3D Mesh Segmentation". In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 28.3 (Aug. 2009).

[SC17] Rohan Sawhney and Keenan Crane. "Boundary First Flattening". In: *ACM Trans. Graph.* 37.1 (Dec. 2017), 5:1–5:14. ISSN: 0730-0301. DOI: 10.1145/3132705. URL: http://doi.acm.org/10.1145/3132705.

[Cra18] Keenan Crane. "Discrete Conformal Geometry". In: *Proceedings of Symposia in Applied Mathematics*. American Mathematical Society, 2018.

[LVS18] Max Limper, Nicholas Vining, and Alla Sheffer. "BoxCutter: Atlas Refinement for Efficient Packing via Void Elimination". In: *ACM Transaction on Graphics* 37.4 (2018). DOI: 10.1145/3197517.3201328.

[SSC18] Yousuf Soliman, Dejan Slepčev, and Keenan Crane. "Optimal Cone Singularities for Conformal Flattening". In: *ACM Trans. Graph.* 37.4 (2018).

[OF] Joe O'Rourke and Komei Fukuda. *Unfolding convex polytopes*. Available at http://jeffe.cs.illinois.edu/open/unfold.html (11/01/2018).