Alpha Hulls

Michael Hemmer, Cédric Portaneri, Pierre Alliez

Alpha Hulls

Michael Hemmer (Google), Cédric Portaneri (Inria), Pierre Alliez (Inria)

December 2, 2020

1 Context

Accurate and valid 3D digital representations are required for digital twins of production facilities and robotic applications. In this project we consider settings where the input 3D geometry is limited to boundary representations in the form of triangle soups, defect-laden and with overly high resolution. We address the problem of converting such defect-laden inputs into valid solid representations with just-enough details for enabling efficient processing.

Defects. The input 3D geometry that needs to be processed is often overly detailed yet imprecise and invalid. More specifically, the input triangle soups are either automatically generated from CAD models or reconstructed from laser scanned point sets. Most of these models fall short on providing a consistent (valid) boundary representation of the physical real world objects they are supposed to model. Beyond outliers, such as missing or ill-placed triangles, the most common defects are open boundary edges of triangular patches that do not match their corresponding counterparts, which results in gaps, islands or self intersections.

While this can be tolerated to some extent for collision detection in which the geometric primitives are anyway organized in an acceleration data structure such as a bounding volume hierarchy, grid or octree, it is very inconvenient to have an inconsistent boundary representation. For instance, the notion of exterior and interior of an object are inherently ill-defined and tasks such as "remove inner geometry", "provide a conservative simplification", "compute thickness" or "swept volume computations" are either a gamble and/or are significantly slowed down.

Aiming for an industrial product at Google scale it is extremely important to stress the fundamental role that the geometry processing will play for our project/Ballet. Considering that the problem definition of one cell from the recent pilot had around 45k geometric objects it is clear that even a seemingly low failure rate of just 0.01% would impact the scalability of the product. Therefore, in order to achieve robustness, the geometric processing package has been refactored to follow the Exact Geometric Computing Paradigm in order to eliminate additional uncertainty that would be introduced by solely implemented algorithms using error prone floating point arithmetic.

Requirements. We have identified two crucial geometric processing tasks for our project. Since the meshes that are given to us are not precise and topological inconsistent, we need an operator that converts those into valid solids, that is, each solid object must be represented by a closed triangular mesh that is free of self intersections (and therefore orientable). Such valid 3D models offers a well-defined notion of inside and outside and allows further processing. In addition, the resulting mesh must strictly contain the input (i.e., be conservative) so that we can guarantee that we are not violating any constraints (such as safety margins). In addition, we must remain tight to the input, that is, we should not add extra material everywhere, as this would make the robot motion planning problem harder and in the worst case even infeasible.

In short, we require a tight solid outer approximation of the input. We define next these three notions.

- Tight. By tight we express that we can not afford to achieve this by, e.g., simply providing a uniform offset surface of some large epsilon as this would make the underlying robot motion planning problem much harder. Inflating objects would shrink the free configuration space and in extreme cases render the motion planning problem infeasible.

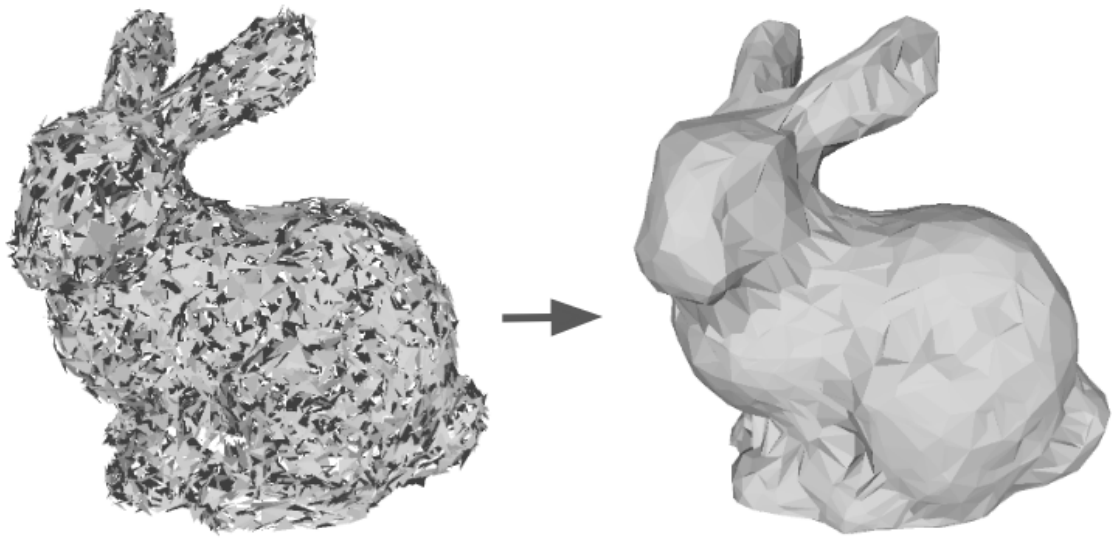

Figure 1: Geometry sanitizer. Left: defact-laden bunny. Right: output alpha hull.

- Solid. In the literature a solid is commonly referred to as a shape that could also exist in the physical real world. Specifically, it must be bounded by a closed surface, have no low dimensional features and provide a well-defined notion of interior and exterior. Thus, a valid boundary representation (usually a surface mesh) conforming to such a solid definition is a closed 2-manifold that is free of self intersections.
- Outer Approximation. Outer approximation (also commonly referred to as conservative approximation) implies that the resulting solid object must strictly contain the original input triangle soup. Such a conservative approach ensures that the robots never collides the physical objects in order to obey all safety requirements.

2 Related Work

The quest for conservative approximations has generated a series of contributions [8, 5, 6, 3, 11, 4, 2, 1, 9], and converting a triangle soup from the wild into a valid solid approximation even without further guarantees is still an active area of research [7].

The approach entitled "Tetrahedral Meshing in the Wild" compares with three algorithms currently available in the CGAL library [10]. However, and while the results are indeed impressive from the robustness point of view this approach has one fundamental flaw. It uses winding numbers to eventually define which cells of the 3-dimensional triangulation are considered inside and outside. The notion of winding number utilized in this approach is inherently a heuristic, which is not appropriate for our context as there is no guarantee that the result contains the input.

3 Alpha Hull

The term alpha hull has first been introduced by Herbert Edelsbrunner. The alpha hull as defined in the PhD manuscript of Herbert Edelsbrunner is the complement of the union of all open balls of radius alpha that do not contain a point of the input. The alpha shape is considered as the linearization of this shape. The term is still used in the context of generalizations of the alpha shape, that is, for weighted input points to for instance approximate the shape of molecules.

The alpha hull has certainly been studied before as a mathematical object. This is not a surprise as it is a rather straightforward generalization of the convex hull. The convex hull, in short, is the closure

of the union of all segments defined by all possible pairs of points from the input, where closure means that one repeats this process until the set does not change anymore. E.g. for an input in 3D, one may need up to three rounds. The alpha hull is a generalization of the convex hull in the sense that it is defined as the union of all convex hulls of the input within balls of radius alpha.

Definition. For an input set S, the alpha hull with scalar parameter alpha, in short AH(S,alpha), is defined as the infinite union of all convex hulls of the input within a ball of radius alpha

$$AlphaHull(S, \alpha) = \bigcup_{\forall_{x \in \mathbb{R}^3}} (CH(Ball(x, \alpha) \cap S))$$

Remarks

- The definition is not constructive as there are as many balls as there are points in R3.
- The alpha hull is not defined as a closure.
- The alpha Hull is not a simplicial complex, i.e. not composed solely of points, line segments and triangles.
- A major flaw of alpha hulls which makes them rather inconvenient to handle is that they induce concave surface patches in concave parts of the input object. That is, even though the input is a set of triangles, the operator leaves the realm of surfaces that can be again represented by triangular meshes (aka simplicial complexes).
- The Alpha Hull is not idempotent as in general, the alpha hull of an input alpha hull and the input alpha hull are not the same. Specifically, the closure of the alpha hull operator results in a set of convex hulls.

Alpha hull vs alpha shape. The notion of Alpha Hull should not be confused with the one of the Alpha Shape, which is only defined for a point set P. The Alpha Shape is conveniently defined on top of the Delaunay Triangulation of P, as the set triangles of the Delaunay Triangulation whose vertices can be touched by a ball of radius alpha while not actually containing any of the points in P, including the three points of the triangle which must lie on the boundary of the ball. While not going into details this can lead to configurations where, for a point set P, a triangle that fits into a ball with radius alpha (and therefore is part of the Alpha Hull) is not part of the Alpha Shape. We are not aware of a generalization of alpha shapes to inputs other than points as its definition is based on Delaunay Triangulation, which is defined for point sets only.

4 Implementation

We implemented the approximate alpha hull algorithm using the CGAL library and exact arithmetic. Our algorithm takes a input a surface triangle mesh and a user-defined scalar alpha value. There is no prerequisite on the input connectivity so that it can take an arbitrary triangle soup, with self-intersections. It generates as output either a watertight surface triangle mesh whose interior volume strictly encloses the input triangle soup, or a message stating that the input is not alpha valid, in the sense that it contains holes larger than alpha.

Algorithm. The algorithm starts by inserting all input vertices into a 3D Delaunay triangulation. It then performs an initial point sampling through recursive longest edge bisection applied to the facets of the input triangle soup, for which the smallest enclosing sphere radius is larger than alpha/2, and as long as the facet is not already represented in the Delaunay triangulation. All those new sample points are added to the Delaunay triangulation. In the Delaunay triangulation from the CGAL library, all triangle facets are adjacent to two tetrahedron cells. Each facet of the boundary of the Delaunay triangulation, which coincides with one facet of the convex hull of the triangulation vertices, is adjacent

to one so-called infinite tetrahedron cell, an abstract cell connected to a so-called infinite vertex to ensure the aforementioned double facet adjacency.

The algorithm then proceeds to flood fill, which consists of a BFS traversal of the cells of the Delaunay triangulation. Initially, all cells are tagged as inside except the infinite cells that are tagged outside. Flood filling consists of traversing the cells via the adjacency graph between cells, starting from the infinite ones tagged as outside, and tagging the traversed cells as outside whenever possible.

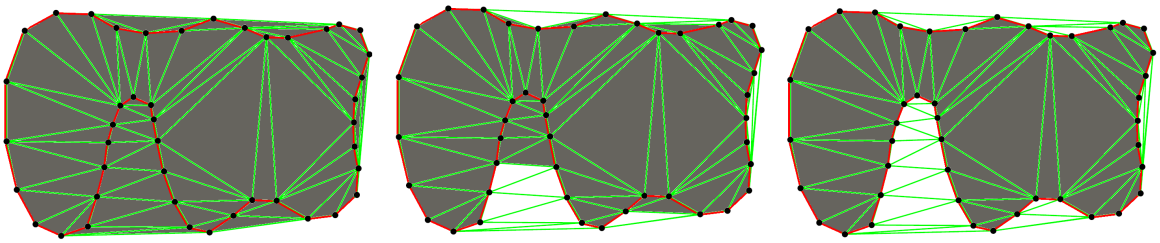

Figure 2: Flood fill algorithm. from left to right. Special case where the Delaunay triangulation is conforming.

Given a cell and its neighbor cell, we can visit its neighbor if the facet between them is not part of the input and if its smallest enclosing sphere is larger than alpha. Once the neighbor cell is visited, we tag it as outside and resume traversal only when the neighbor cell does not intersect the input triangle soup in a non-conforming way. If it does, we insert a sample point into the Delaunay triangulation at the intersection between this neighbor cell and the input to trigger a cell removal through Delaunay insertion. We resume the flood fill considering the newly created cells.

In order to know whether a Delaunay facet is part of the input triangle soup, or to know whether the neighbor cell is intersecting the input in a non conforming way, we devised a specific data structure that represents the input. For each set of coplanar faces of the input, we construct a constrained 2D Delaunay triangulation in 3D where each vertex is mapped to the 3D Delaunay triangulation. Because the two types of Delaunay triangulation have most of the time the same connectivity locally, we can efficiently check whether a 3D facet exists or whether a 3D cell is conforming in this space of 2D triangulations.

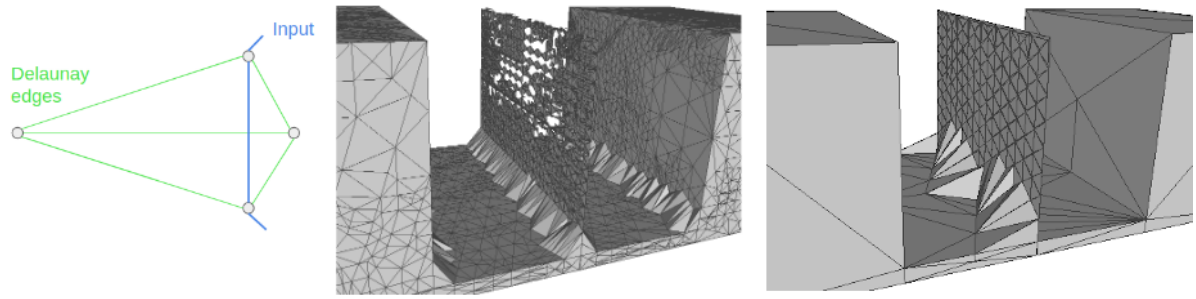

Figure 3: Dealing with non conforming cases. Left: given some constrained edges (blue), a Delaunay triangulation is non-conforming when its edge (green) does not cover all the constraints. Right: case where the volume of the input geometry in the middle is so thin that it is not represented in the Delaunay triangulation. In this case the flood fill tags both adjacent cells of those non represented facets as exterior.

Output. The output is a surface mesh that represents the outer boundary of the hull. It has the same properties as the hull itself as it contains the input and is bounding a volume. It is tight in convex areas and has spurious volume elements in concave areas. The Hausdorff distance from the output mesh to

the input is at most equal to the alpha value. The output is combinatorially manifold as it is extracted from the Delaunay triangulation, but it can contain non-manifold edges, always adjacent to an even number of face. The output mesh geometry is represented via exact numbers in order to ensure all above properties. In theory, converting such numbers to double precision numbers might yield self-intersections and a volume not containing the input anymore.

```
// Initial Sampling
Insert input vertices in Delaunay Triangulation
While sample points are generated
  For each input face
    If face is not represented in Delaunay and big
    enough (smallest enclosing sphere radius > alpha/2)
      Generate sample point : Longest Edge Bisection
  Insert all sample points in Delaunay
// Alpha Flood Fill
While sample points are generated
  Tag all Delaunay cells as inside except infinite cells
  Insert infinite cells inside a queue
  While queue is not empty
    Get cell
    For each finite facet of the cell
      If is traversable, e.g. : not input face and big
      enough (smallest enclosing sphere radius > alpha)
        If the neighbor cell intersect the input
        (non conforming intersection)
          Generate sample point on the intersected input
          face to destroy cell
        Else
          Set the neighbor cell outside
          Add the neighbor cell in the queue
  Insert all sample points in Delaunay
```

Figure 4: Pseudo-code of the alpha hull algorithm.

References

[1] H. Borouchaki and P.J. Frey. Simplification of surface mesh using hausdorff envelope. *Computer Methods in Applied Mechanics and Engineering*, 194(48-49):4864 – 4884, 2005.

[2] Mario Botsch, David Bommes, Christoph Vogel, and Leif Kobbelt. GPU-based tolerance volumes for mesh processing. In *Pacific Conference on Computer Graphics and Applications*, pages 237–243. IEEE Computer Society, 2004.

[3] A. Ciampalini, Paolo Cignoni, Claudio Montani, and Roberto Scopigno. Multiresolution decimation based on global error. *The Visual Computer*, 13(5):228–246, 1997.

[4] Jonathan Cohen, Dinesh Manocha, and Marc Olano. Successive mappings: An approach to polygonal mesh simplification with guaranteed error bounds. *International Journal of Computational Geometry and Applications*, 13(1):61–96, 2003.

[5] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification envelopes. In *Proceedings of ACM Conference on Computer Graphics and Interactive Techniques*, pages 119–128, 1996.

[6] André Guéziec. Surface simplification inside a tolerance volume. Technical Report 20440, 1996. IBM Research Report RC 20440.

[7] Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. Tetrahedral meshing in the wild. *ACM Trans. Graph.*, 37(4):60:1–60:14, July 2018.

[8] Alan D. Kalvin and Russel H. Taylor. Superfaces: Polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Applications*, 16(3), 1996.

[9] Manish Mandad, David Cohen-Steiner, and Pierre Alliez. Isotopic Approximation within a Tolerance Volume. *ACM Transactions on Graphics*, 34(4):12, 2015.

[10] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.1.1 edition, 2020.

[11] Steve Zelinka and Michael Garland. Permission grids: Practical, error-bounded simplification. *ACM Transactions on Graphics*, 21(2):207–229, 2002.

Contents