r3d: Software for fast, robust geometric operations in 3D and 2D

LA-UR-15-26964

Devon Powell

31 August 2015

Abstract

We describe updates to the software, r3d, introduced in Powell and Abel (2015). The original published version of r3d provided functions to robustly clip an axis-aligned cube against the four faces of a tetrahedron and to then calculate the moments over the resulting polyhedron. We referred to this as "physically conservative voxelization," which is a direct remap from a set of tetrahedral cells onto a regular Cartesian grid. The scheme itself relies on the combinatorial abstraction of Sugihara (1994) to achieve robustness against degenerate geometry, and is applicable to general convex polyhedral intersections.

In this report, we document recent additions and improvements to the r3d software. The biggest change was to extend the API beyond the special case of voxelization to make intersections between general polyhedral mesh cells as straightforward as possible. We also modified the clipping algorithm to be able to handle nonconvex polyhedra. The code for computing geometric moments was replaced by the fast recursive method of Koehl (2012). Other minor changes are described in the main text.

In addition, we have added a beta version of a general N-dimensional version of the software to intersect polytopes in arbitrarily high-dimensional spaces. This software, rNd, is experimental and is still under development at this time.

A full set of unit tests for robustness, speed, and accuracy was added as well. These tests serve both as a tool for test-driven development and as a set of examples for common use cases.

1 Introduction

We are concerned with the problem of robustly truncating a polyhedron against a plane, the so-called "clipping and capping" problem. This is the basic operation underlying any algorithm that intersects one polyhedron with another convex polyhedron, since any convex polyhedron can be represented as an intersection of half-spaces. This in turn forms the kernel for a general direct remeshing scheme, since the intersection volume between two polyhedral cells can be constructed and the resulting moments computed.

This report is an update to the previous work described in Powell and Abel (2015), in which an algorithm based on that of Sugihara (1994) is applied to the exactly conservative remapping of tetrahedral cells to a Cartesian grid. The purpose of the associated software, r3d ("Robust 3D"), is to provide a unified framework for quickly and robustly constructing the intersection volumes between convex polyhedra ("clipping") and subsequently computing the moments over those volumes ("reduction"). Since the publication of Powell and Abel (2015), we have extended and modified r3d in several ways:

1. We have added capability for clipping nonconvex and multiply-connected polyhedra. See Section 3.2 for a discussion of the importance of this modification.

2. We have implemented the fast, recursive method of Koehl (2012) for computing arbitrary geometric moments of polyhedra in optimal time.

3. We have added several utility functions for creating, transforming, and checking the topological validity of polyhedra.

4. We have made some algorithmic simplifications that make the code cleaner and more readable.

5. We have modified the API to be more easily applied to the general remeshing case.

6. We have added a unit-testing suite for verifying the speed and robustness of the code.

7. We have implemented a beta version of `rNd`, a generalization of the clipping and reduction operations to polytopes in arbitrary dimensions.

The remainder of this report is laid out as follows. First, we give a brief review of the concept of combinatorial abstraction and its application to fast, robust polyhedral intersections. We then describe the algorithm and some differences from the original version. We finally give a brief overview of the current capabilities of `r3d`, including the new unit-testing code.

2 Combinatoric abstraction and robustness

The subject of geometric robustness in computational geometry has been well-studied in the literature. We previously explored the subject in Powell and Abel (2015), relying heavily on a review by Stewart (1994). For our purposes, the method of combinatorial abstraction proposed by Sugihara proved to be a simple, elegant solution to the robustness problem. We give a brief description here.

Combinatoric abstraction, first addressed in this context by Sugihara (1994), refers to the design of a clipping and capping algorithm that places the topological validity of the output above all else. Specifically, the algorithm represents simple polyhedra using their edge-vertex graphs (their "1-skeleta") and operates on them in such a way that the topological validity of the output is guaranteed. Numerical comparisons, which are of finite precision, are of secondary importance and used only as a guide for combinatoric operations on the edge-vertex graph.

This is an exploitation of several theorems in polyhedral combinatorics. First, Balinski's theorem (Balinski, 1961) states that the edge-vertex graph of a N-dimensional polytope is at least N-vertex-connected (exactly N-vertex-connected for a simple polytope). A converse of this statement, proven by Blind and Mani-Levitska (1987) and Kalai (1988), is that the combinatoric structure (the adjacencies of faces, edges, etc.) for a simple polytope is uniquely determined by its edge-vertex graph. Another partial converse that we will use for illustration purposes is Steinitz' theorem (Steinitz 1922), which states that any convex polyhedron in 3 dimensions can be represented using a 3-vertex-connected planar graph (this is called a Schlegel diagram). See Figure 1 for an illustration.

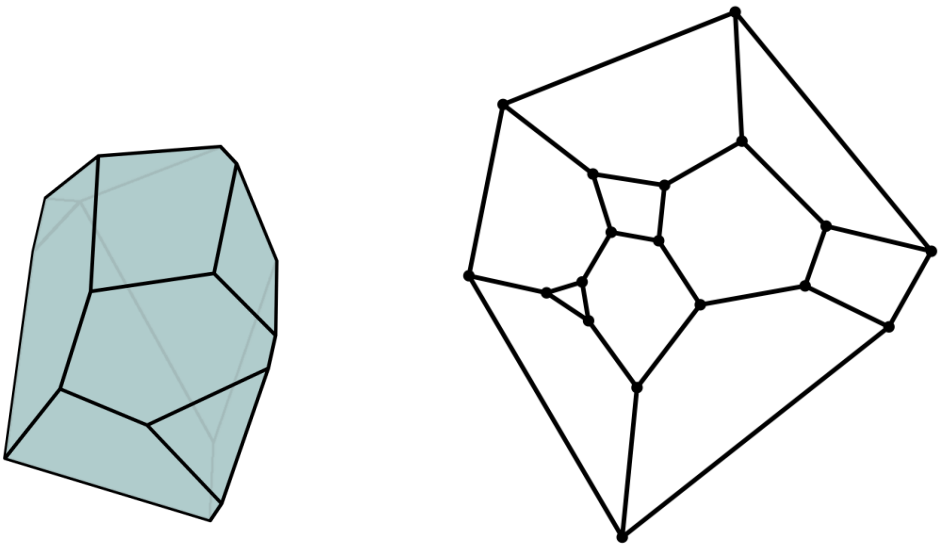

Figure 1: A 3D polyhedron (left) and its 3-vertex-connected planar graph, or Schlegel diagram (right). The outer boundary of the Schlegel diagram also corresponds to a face of the polyhedron.

These theorems together guarantee that, given the proper algorithm for modifying the edge-vertex graph of a simple polyhedron, the output will be a polyhedron that is both valid and unique, and hence always correct. This allows us to implement a simple clipping and capping algorithm that is robust

yet does not rely on any tolerances, edge cases, exact arithmetic, etc. Rather, we inherit geometric robustness from this combinatoric abstraction because we use numeric evaluations only as a guide for combinatoric operations, which insulates us from the usual robustness issues that plague computational geometry.

3 Algorithm

3.1 Clipping and capping

In abstract terms, we are carrying out a criterion-based graph partitioning. We identify the connected components of the edge-vertex graph (henceforth "the graph") whose vertices lie in front of the clip plane, then insert new edges around the boundary of each connected component to form a new face. Sugihara gives a proof that such an algorithm preserves topological validity.

We store polyhedra as triply-linked sets of vertices, where each vertex v contains a position $\mathbf{x}$ and three pointers to the neighboring vertices. These pointers are ordered such that the three edges emanating from a vertex always appear in counterclockwise order (clockwise would work just as well; this is simply a matter of convention) when viewed from outside the polyhedron. This allows us to walk over edges of the graph in the proper order. Note that we are restricted to simple polyhedra (strictly three edges per vertex), but we can emulate more that three edges per vertex using multiple degenerate vertices linked together.

The only geometric comparison we need for the clipping procedure is the signed distance between a plane and a point. All clip planes in `r3d` are specified by a unit normal $\hat{\mathbf{n}}$ and a distance to the origin s, such that the signed distance from a vertex v at position $\mathbf{x}$ to the clip plane is given by $d_v = \hat{\mathbf{n}} \cdot \mathbf{x} + s$. By our convention, any vertices with $d_v \geq 0$ are considered to be "in front of" the clip plane, and any vertices with $d_v < 0$ are "behind" the clip plane.

A high-level description of the clipping and capping procedure is as follows:

1. For each vertex v, calculate and save the signed distance d_v from v to the clip plane.
2. For each edge that crosses the clip plane (has one vertex v_0 with $d_0 \geq 0$ and one vertex v_1 with $d_1 < 0$), create a new vertex v_n. Doubly link v_n with v_0. Calculate the position of v_n using the weighted average $\mathbf{x}_n = (d_0\mathbf{x}_1 - d_1\mathbf{x}_0)/(d_0 - d_1)$.
3. For each newly created vertex v_n, walk around edges of the graph in a counterclockwise fashion until another new vertex v_m is reached. Doubly link v_n with v_m, forming a new edge.
4. Remove all vertices with $d_v < 0$.

This is a modification of the depth-first graph traversal proposed in Powell and Abel (2015), and it leads to simpler, more readable code, as well as the ability to clip nonconvex polyhedra. For an illustration of this clipping process, see Figures 2 and 3. This clipping and capping algorithm is implemented in the function `r3d_clip()`.

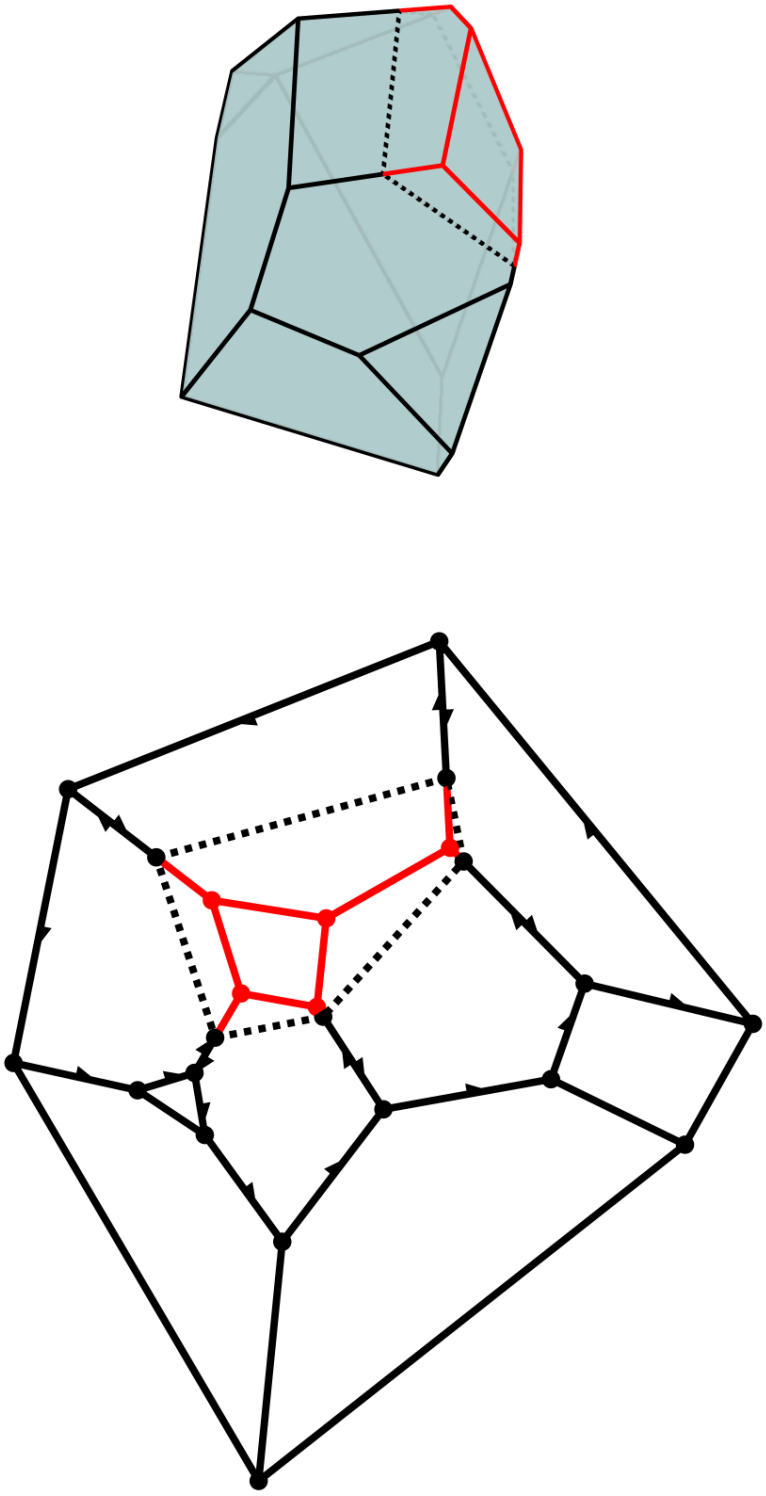

Figure 2: The clipping process, illustrating edges and vertices to be clipped (red) and new edges and vertices to be inserted (dashed black). Arrows indicating the graph traversal order for inserting new edges and vertices are shown in the Schlegel diagram (bottom).

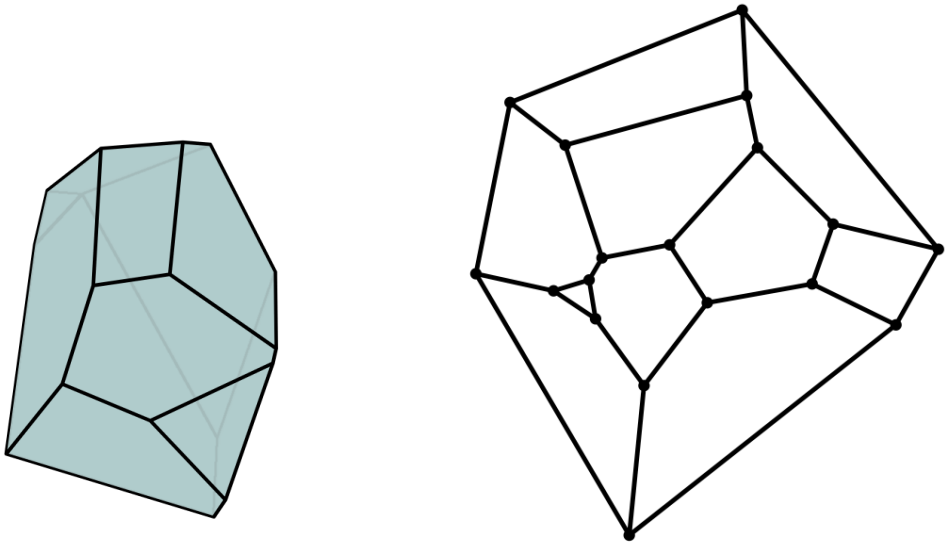

Figure 3: The clipped polyhedron.

3.2 A note on convexity

The algorithm is largely the same as that presented in Sugihara (1994), save for a subtle difference. As Sugihara is mainly interested in the topological validity of the output, they only address the case of convex polyhedra. Indeed, Powell and Abel (2015) assume that this is the case as well, a practical assumption for clipping axis-aligned cubes.

However, finite-precision representation of vertex coordinates can lead general polyhedra that are macroscopically convex to have numerical nonconvexities on the order of machine precision. The result is that in a nonconvex polyhedron there can be more than one connected component of the graph that lies in front of the clip plane.

As such, it is important for our algorithm to be able to handle convex/nonconvex intersections, even when the polyhedral cells involved are nominally convex. The algorithm presented here is a slight modification to the one presented in Powell and Abel (2015), and it naturally handles nonconvex polyhedra as well as multiply-connected polyhedra and polyhedra with holes (Euler characteristic $\chi \neq 2$).

One important thing to note is that when inserting new edges, the algorithm must always walk around the component of the graph that is inside the clip plane. Walking around faces outside of the clip plane in order to add new edges to the component inside the clip plane can lead to non-3-vertex-connectedness and hence an invalid polyhedron. For example, this can occur when there are two disjoint components that lie inside the clip plane and also share a face of the original polyhedron.

The discussion of numerical nonconvexity is a good place to reiterate the nature of this algorithm. Namely, the clipping algorithm operates solely on a 3-vertex-connected graph whose vertices have each been assigned a signed number d_v representing their distances to a clip plane. It is agnostic to the notion of vertices on a face being "coplanar" in an exact mathematical sense. What *is* guaranteed is that the output is always another unique, valid edge-vertex representation of a polyhedron. The notion of "topological correctness" hence does not apply here, since for the output to be topologically correct, the input must also be, which can never be guaranteed in finite precision. However, any topological incorrectness in the input must be on the order of machine precision and hence any resulting effect on the final computed moments of the output will be negligible.

3.3 Calculation of moments

Another modification to the code comes in our implementation of the fast recursive algorithm of Koehl (2012) for calculating the moments of a tetrahedron. Successively higher-order moments are computed from the lower-order moments, giving an algorithm that is $\mathcal{O}(n^3)$ in the polynomial order, which is optimal. Our edge-vertex representation of polyhedra allows for a natural decomposition of each face

pyramid into a fan of tetrahedra (see Figure 4), and it is over this set of simplices that we compute moments.

This moments calculation is implemented in the function `r3d_reduce()`.

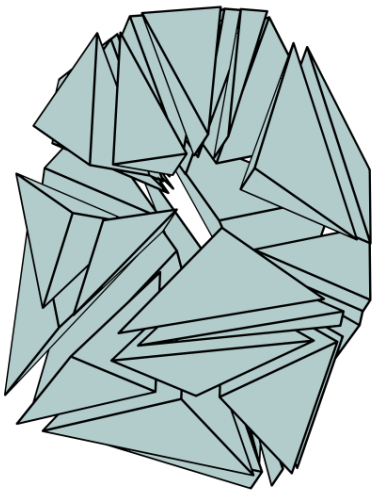

Figure 4: The simplicial decomposition used for calculating moments.

4 Features of `r3d`

4.1 Functions

The functions `r3d_clip()` and `r3d_reduce()` form the core of the `r3d` API. However, there are several auxiliary functions provided to make `r3d` as user-friendly as possible.

`r3d_init_tet()`, `r3d_init_box()`, and `r3d_init_poly()` allow the user to create a polyhedron in `r3d`'s native edge-vertex graph representation by providing the desired vertex coordinates. `r3d_orient()` lets the user check four vertices to ensure that a tetrahedron formed with the vertices in that order will have positive volume. `r3d_init_poly()` is the most general, taking a list of vertex coordinates, as well as lists of vertex indices as they appear in counterclockwise order (as viewed from the outside) around each face. The user can, of course, always initialize a polyhedron using their own code, though they must guarantee that it is valid for subsequent `r3d` calls to work as expected.

Likewise, `r3d_tet_faces_from_verts()`, `r3d_box_faces_from_verts()`, and `r3d_poly_faces_from_verts()` are the analogous functions for creating a set of planes from the faces of the provided polyhedra. These planes can then be passed to `r3d_clip()` in order to intersect two polyhedra.

The functions `r3d_rotate()`, `r3d_translate()`, `r3d_scale()`, `r3d_shear()`, and `r3d_affine()` allow the user to apply linear transformations to a polyhedron. `r3d_affine()` is the most general, providing a full 4×4 homogeneous matrix transformation.

Two functions, `r3d_is_good()` and `r3d_print()`, are provided to aid the user in debugging code. `r3d_is_good()` checks that the polyhedron's edge-vertex graph is indeed 3-vertex-connected and satisfies Balinski's theorem. This is a costly operation, so the function is unsuitable for inclusion in performance code. `r3d_print()` simply dumps the vertex coordinates and connectivity.

In `r3d`, polyhedra are structs (`r3d_poly`) with a finite-sized vertex buffer. The capacity of this buffer can be set with the macro `R3D_MAX_VERTS` in `r3d.h`. The default, 128, is plenty for computational physics applications and the like. In the event of a segfault, a usual culprit is `R3D_MAX_VERTS` being too small. The user can also potentially speed the code by minimizing `R3D_MAX_VERTS` for a specific application to allow for better data locality.

All detailed documentation is provided in Doxygen format in the header files `r3d.h` and `r2d.h`.

4.2 r2d

In addition to r3d, we also provide analogous 2D versions of all functions with the prefix r2d. Some functions have slightly different names to reflect the lower dimensionality (e.g. "r2d_init_tri()" vs. "r3d_init_tet()"), but their purpose should still be obvious. All algorithms discussed here extend trivially to the 2D case.

4.3 rNd

We have also implemented the clipping and reduction algorithms for general N-dimensional polytopes and hyperplanes under the prefix rNd. Documentation is in rNd.h. This code is still under development and has not been rigorously tested, so it is not guaranteed to work properly at this time.

4.4 Unit-testing framework

We also provide several unit tests, which reside in tests/r3d_unit_tests.c and tests/r2d_unit_tests.c. While these tests are for verifying that the code works as advertised, they also serve as good examples of how to use the r3d API.

We highly encourage the user to begin by looking at test_split_tet_thru_centroid(), which shows the simple base case of initializing a tetrahedron, clipping it against a plane, and computing its moments.

The other tests that serve as good use case examples are as follows: test_split_nonconvex() demonstrates that r3d gives correct results, even when the input is nonconvex, by splitting apart a "zig-zag prism" along the three axes. It also shows how the user can initialize a polyhedron without using one of the r3d_init functions. test_tet_tet_timing() generates pairs of tetrahedra and intersects them. Each pair of tetrahedra takes $\sim 8\mu$s to process. This also demonstrates how one should use the code in an actual remeshing scheme between two polyhedral cells. test_torus_load_and_chop() creates a torus using r3d_init_poly() and then clips it, demonstrating the ability of r3d to handle nontrivial polyhedra. Finally, test_moments() demonstrates how to calculate moments up to arbitrary order and how to access them in the correct order upon return.

The most rigorous unit tests are the ones prefixed "test_recursive_splitting". These initialize a tetrahedron and split it along a cutting plane that is geometrically degenerate with the tetrahedron in some way. The moments for the two halves are computed and compared to ensure that they are consistent with the original. This process goes on recursively until a user-specified depth is reached.

5 Conclusion

We have updated the r3d software beyond the special case of remeshing tetrahedral cells to a Cartesian grid, as was originally described in Powell and Abel (2015).

r3d is now a complete and user-friendly framework for computing intersections between two convex polyhedra, or between one nonconvex and one convex polyhedron. The clipping operation is geometrically robust and fast. The reduction operation is now based on the algorithm of Koehl (2012), and computes moments to arbitrarily high order in optimal time. A number of support functions allow the user to convert between different descriptions of polyhedra, apply affine transformations, and check for topological validity.

References

Balinski, M. L. (1961). On the graph structure of convex polyhedra in n-space. *Pacific J. Math. 11*(2), 431–434.

Blind, R. and P. Mani-Levitska (1987). Puzzles and polytope isomorphisms. *Aequationes Math. 34*(2-3), 287–297.

Kalai, G. (1988). A simple way to tell a simple polytope from its graph. *J. Combin. Theory Ser. A 49*(2), 381–383.

Koehl, P. (2012). Fast recursive computation of 3d geometric moments from surface meshes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*(11), 2158–2163.

Powell, D. and T. Abel (2015). An exact general remeshing scheme applied to physically conservative voxelization. *Journal of Computational Physics 297*, 340 – 356.

Steinitz, E. (1922). *Polyeder und Raumeinteilungen*, Volume 3 of *Encyclopädie der mathematischen Wissenschaften*, pp. 1–139. B.G. Teubner Verlag.

Stewart, A. J. (1994). Local robustness and its application to polyhedral intersection. *International Journal of Computational Geometry and Applications 4*(1), 87–118.

Sugihara, K. (1994). A robust and consistent algorithm for intersecting convex polyhedra. *Computer Graphics Forum 13*(3), 45–54.