Barycentric Interpolation on the 7-Sphere

Sterling G. Baird[a,*], Eric R. Homer[a], David T. Fullwood[a], Oliver K. Johnson[a]

[a] *Department of Mechanical Engineering, Brigham Young University, Provo, UT 84602, USA*

Abstract

We present a method for performing efficient barycentric interpolation for large point sets which reside on the surface of less than a hemisphere of a hypersphere. This method includes removal of degenerate dimensions via singular value decomposition (SVD) transformations, determination of intersecting facets via nearest neighbor (NN) searches, and interpolation. Corresponding MATLAB code is available at `github.com/sgbaird-5dof/interp`.

Keywords: barycentric, interpolation, hypersphere, octonion, triangulation

1. Introduction

Barycentric coordinates are a type of homogeneous coordinate system that reference a prediction point within a simplex [1] or convex polytope [1–3] based on "masses" or weights at the vertices, which can be negative. The prediction point is assumed to be the barycenter (center of mass) of the simplex or convex polytope, and weights at the vertices necessary to make this assumption true are determined. We utilize rigid SVD transformations and a standard triangulation algorithm (quickhull [4] via `delaunayn()` in `sphconvhulln.m` to define a simplicial mesh (Section 2). We then use barycentric weights (i.e. coordinates) for computing intersections of a point within a simplicial facet (Section 3) and for interpolation (Section 4) [1]. For further information on barycentric coordinates and its applications and generalizations, see [1–3, 5–18].

Built-in MATLAB functions are indicated with parantheses (e.g. `delaunayn()`), whereas custom functions are indicated with the `.m` extension (e.g. `sphconvhulln.m`).

2. Triangulating a Mesh

Creation of a simplicial mesh is necessary to perform barycentric interpolation. Due to the difficulty of visualizing a 7-sphere, we provide visual illustrations of the process as applied to lower-dimensional analogues. The triangulation process occurs by:

1.1 applying a SVD transformation to remove any degeneracies (e.g. U(1)-symmetry degeneracy for grain boundary octonions [19]) inherent in the coordinates (Section 2.1)

1.2 When the data resides on less than a hemisphere, linearly projecting points onto a hyperplane that is tangent to the vector between the origin and the mean of the input points to reduce computational burden of the triangulation (Section 2.2)

*Corresponding author.
Email address: `ster.g.baird@gmail.com` (Sterling G. Baird)

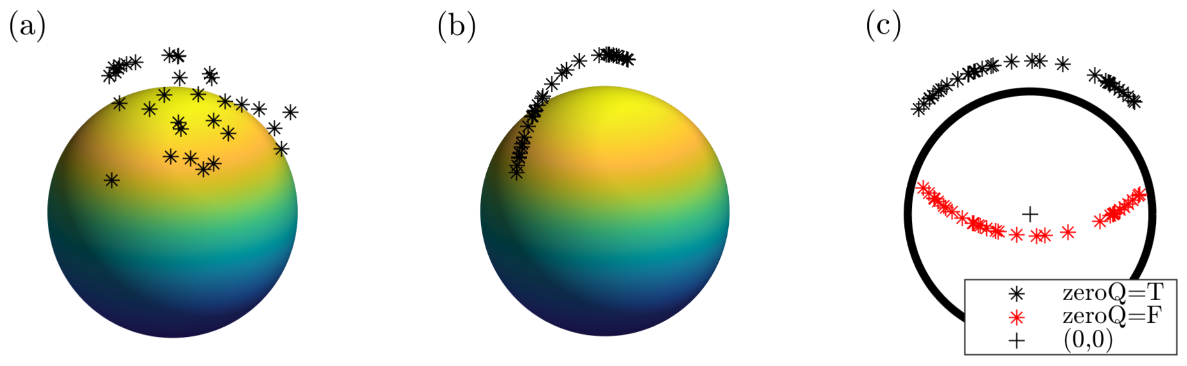

Figure 1: 3D Cartesian to 2D Cartesian analogue of 8D Cartesian to 7D Cartesian degeneracy removal via rigid SVD transformation as used in barycentric interpolation approach. (a) Starting spherical arc points on surface of 2-sphere, (b) rotational symmetrization applied w.r.t. z-axis (analogous to U(1) symmetrization), and (c) degenerate dimension removed via singular value decomposition transformation to 2D Cartesian with either the origin (black plus) preserved (black asterisks, `zeroQ=T`) for triangulation or ignored (red asterisks, `zeroQ=F`) for mesh intersection. The spheres (a,b) and circle (c) each have a radius of 0.8 and are used as a visualization aid only.

1.3 performing a second SVD transformation (Section 2.3)

1.4 computing the triangulation according to the quickhull algorithm [4] using built-in methods

In the explanation of each step that follows, we make reference to lower-dimensional visual analogues of the triangulation procedure, which are given in Figure 1, Figure 2, and Figure 3. We note that 3D Cartesian coordinates in Figure 1 correspond to 8D Cartesian coordinates, whereas 3D Cartesian coordinates in Figure 2 and Figure 3 correspond to 7D Cartesian coordinates. This is intentional for two reasons.

First, Figure 1 illustrates that 8D Cartesian points constrained to the surface of a hypersphere are analogous to a point cloud on the 2-sphere (Figure 1a) and that an 8D Cartesian point set constrained to the surface of a hypersphere is analogous to a geodesic arc on the 2-sphere (Figure 1b). If a point set has a degenerate dimension, this can be removed by a rigid SVD transformation to 7D Cartesian coordinates (analogous to 2D Cartesian coordinates in Figure 1c). This sequence would be more difficult to visualize if Figure 1a was meant to represent a point cloud on the 3-sphere (4D Cartesian coordinates), etc.

Second, Figure 2 illustrates a second transformation from normalized 7D Cartesian coordinates (Figure 2a) to a hyperplane (Figure 2b) which is then transformed into 6D Cartesian coordinates via a second SVD. In this case, key issues are retained that would otherwise be lost if an arc on a circle (1-sphere) to 1D Cartesian coordinates were used instead[1]. Additionally, the use of actual triangles is a more familiar and compelling illustration of *triangulation*.

While lower dimensional analogues are useful for visualizing and understanding the process of triangulation, a written description for the full-dimensional space is also given (Sections 2.1–2.3). As appropriate, we refer back to the teaching figures described in this section.

2.1. Singular Value Decomposition Transformation from 8D Cartesian to 7D Cartesian

To reduce the computational complexity of triangulating a high-dimensional mesh [4], some simplifications are made. First, the degenerate dimension (e.g. obtained from analytically minimizing $U(1)$

[1]Non-intersection issues due to high-aspect ratios and consideration of facets connected up to `nnMax` NNs do not manifest in triangulations on the surface of a 1-sphere because one of the two facets (i.e. line segments) connected to the first NN mesh vertex relative to the prediction point is guaranteed to have an intersection.

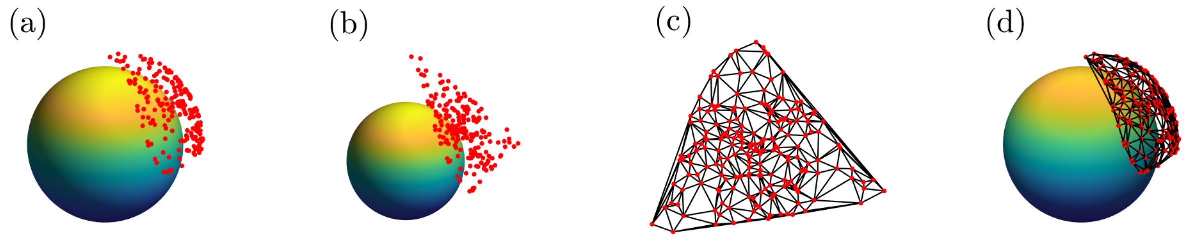

Figure 2: 3D Cartesian to 2D Cartesian analogue of 7D Cartesian to 6D Cartesian mesh triangulation used in barycentric interpolation approach. (a) 3D Cartesian input points are (b) linearly projected onto hyperplane that is tangent to mean of starting points. (c) The degenerate dimension is removed via a rigid SVD transformation to 2D Cartesian and the Delaunay triangulation (black lines) is calculated, with input vertices (red). Delaunay triangulation superimposed onto normalized input points (d). The spheres in (a), (b), and (d) have a radius of 0.8 and are used for visualization only.

symmetry [19]) is removed via a rigid (i.e. distance- and angle-preserving) SVD transformation, analogous to a Cartesian rotation and translation (see 3D to 2D SVD transformation from Figure 1b to Figure 1c).

2.2. Linearly Project onto Hyperplane

Next, the resulting 7D Cartesian representation of each point is projected onto a hyperplane that is tangent to the centroid (i.e. mean) of the point set[2] (Figure 2a). By performing this linear projection, one of the dimensions becomes degenerate.

2.3. Singular Value Decomposition Transformation from 7D Cartesian to 6D Cartesian

This additional degeneracy is removed via a second SVD transformation, this time to 6D Cartesian coordinates (see 3D to 2D projection in Figure 2a-b). Finally, the resulting points can be triangulated via the quickhull algorithm [4] (`sphconvhulln.m` and `delaunayn()`), which relies on Euclidean distances[3]. Because the simplicial mesh is defined by a list of edges between vertices for each simplicial facet, this list applies immediately to the point set in its 7D Cartesian coordinates (i.e. no reverse transformation is necessary to use the mesh on the 6-sphere in 7D).

3. Intersections in a Mesh

Once the triangulation has been determined, we need to find which facet each prediction point intersects (i.e. find the intersecting facet). There are two sub-steps:

2.1 applying the same rigid transformation to the prediction points as was applied to the input points (otherwise the prediction points won't line up properly with the mesh) (Section 3.1)

2.2 identifying facets nearby a prediction point and testing for intersection (Section 3.2).

[2]This is *not* a rigid transformation; however, it approximates one with sufficient accuracy to produce a high-quality triangulation.

[3]While the triangulation algorithm used in this work relies on Euclidean distances, other distance metrics that are non-Euclidean [20] could potentially be incorporated into the barycentric approach such as by doing an edge-length based simplex reconstruction [21, 22] using the triangulation edge lengths.

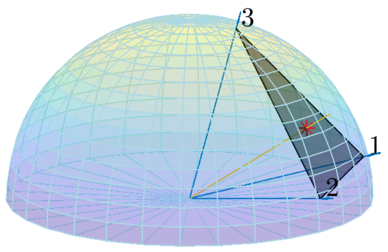

Figure 3: A ray (red line) is linearly projected from the 2-sphere onto the hyperplane of a mesh facet (transparent black), shown as a red asterisk. The barycentric coordinates are computed as $\lambda_{i\in[1,3]} = \frac{1}{3}$. Because all barycentric coordinates are positive, it is determined that the projected point is an intersection with the mesh. Given vertex values of 8.183, 3.446, and 3.188 for vertices 1, 2, and 3, respectively, the interpolated value is calculated as 4.94 via Eq. (1).

3.1. Apply Same Singular Value Decomposition to Input and Prediction Points

The positions of the prediction points need to be fixed relative to the mesh even after the rigid SVD transformation. This is accomplished by:

2.1a concatenating both input and prediction points

2.1b using the `interp5DOF.m` sub-routine `proj_down.m` (via `svd()`) to perform the transformation

2.1c subsequently separating the transformed input and prediction points (reverse of concatenation step)

To map new points onto the mesh, the `usv` structure output from `proj_down.m` needs to be stored and supplied in future calls to `proj_down.m`. Likewise, `usv` need to be supplied to `proj_up.m` to perform the reverse SVD transformation.

3.2. Testing Nearby Facets for Intersections

Once the prediction points are lined up properly with the mesh, the facet containing the prediction point (i.e. intersecting facet) is found. We define the intersecting facet as the one for which a point's barycentric coordinates are positive within a given tolerance. Consequently, we determine facet affiliation by:

2.2a linearly projecting the prediction point onto the hyperplane defined by a mesh facet's vertices (Figure 3)

2.2b computing the point's barycentric coordinates within the facet [16, 23] (`projray2hypersphere.m`)

2.2c testing that all coordinates are positive [1] within a tolerance[4]

2.2d repeating steps 2.2a-2.2c until an intersection is found or a stop condition is reached (see `nnMax` below).

Due to the large number of facets per point of a high-dimensional triangulation (approximately 2000 facets per vertex for a 50 000 point triangulation, or 1×10^8 total facets), some simplifications are made in order to determine intersections of prediction points with the mesh. If every edge length of every

[4]Two tolerances are used: one for the initial computation of barycentric coordinates by projecting onto the hypersphere to determine facet affiliation (`projtol=1e-4`) and a larger tolerance (`inttol=1e-2`) for computation of barycentric coordinates to determine interpolated values (Section 4).

facet were equal, only facets connected to the first NN would need to be considered to find a proper intersection. However, since the points are randomly sampled, edge lengths of facets are non-uniform, and non-unity aspect-ratio facets exist (Figure 2, Figure 4). If the facets have high-aspect ratios, the intersecting facets of prediction points can be far from the NNs mesh points relative to the prediction points (see Figure 4 inset), especially near the perimeter of a hyperspherical surface mesh. Rather than loop through every facet to find an intersection ($\sim 1 \times 10^8$ facets in a 50 000 point mesh), the prediction point intersections are calculated by considering facets connected to up to some number of NN mesh vertices (`nnMax`) relative to each prediction point (in this work, `nnMax=10`). The NN mesh vertices relative to a prediction point are computed (`dsearchn()`). The facet IDs of facets connected to these NNs are computed (via `find()`, as in `find(K==nn)`, where `K` is the triangulation from `sphconvhulln.m` and `nn` is the ID of one of the NN mesh vertices).

Some prediction points will have no intersecting facet found. From our numerical testing, we determine that this non-intersection phenomenon occurs in two situations:

- high-aspect ratio facets (described above)
- prediction points that are positioned just outside the bounds of the mesh but within the bounds of a region, due to the fact that the mesh is a piecewise linear approximation of a surface with a curved perimeter and that randomly sampled points typically do not fall on the true perimeter

In the first case, barycentric interpolation within high-aspect ratio facets may actually lead to worse interpolation error than a NN interpolation strategy due to influence by points far from the prediction point. In the second case, there is no true intersection between the prediction point and the mesh. Both issues can be addressed with the same strategy: we apply a NN approach when an intersecting facet is not found within `nnMax` NNs. In numerical tests, meshes composed of 388 and 50 000 vertices produced non-intersection rates of $(12.07 \pm 1.02)\,\%$ and $(0.68 \pm 0.11)\,\%$, respectively, over approximately 10 trials and using 10 000 prediction points for each trial.

Testing intersections for nearby facets is handled in `intersect_facet.m` and depends on the barycentric coordinate computations in `projray2hypersphere.m`.

4. Interpolation

Once a mesh triangulation has been determined (Section 2), barycentric coordinates are recomputed for a prediction point within the input mesh (Section 3) using a somewhat larger tolerance; the interpolated value is found by taking the dot product of the prediction point's barycentric coordinates and the properties of the corresponding vertices of the intersecting facet via

$$v_{m,q} = \sum_{i=1}^{N} \lambda_{m,i} v_{m,i} \tag{1}$$

where $\lambda_{m,i}$, $v_{m,q}$, $v_{m,i}$ and N, are the barycentric coordinates of the m-th prediction point, interpolated property at the m-th prediction point, property of the i-th vertex of the intersecting facet for the m-th prediction point, and number of vertices in a given facet ($N = 7$ for facets of the simplicial mesh on the degeneracy-free 6-sphere), respectively. Interpolation of many prediction points simultaneously can be accomplished by a simple, vectorized approach (i.e. `dot()` as used in `interp_bary_fast.m`). This function assumes triangulation and weights have been precomputed. In other words, both input and prediction coordinates remain fixed, and only input property values change. If this is the case, barycentric interpolation of new points is incredibly fast. By contrast, if input coordinates change, the

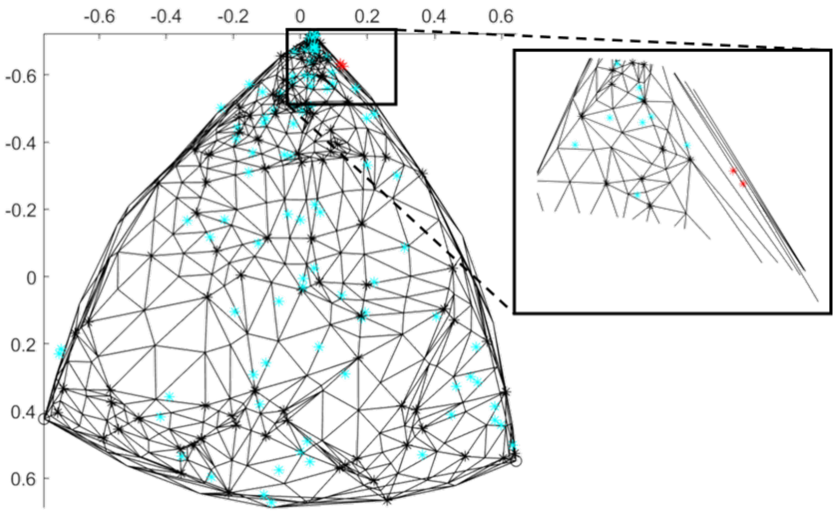

Figure 4: Illustration of two prediction points (red) for which no intersecting facet is found due to being positioned within a high-aspect ratio facet. The inset shows that facets connected to the NN do not contain the prediction point. Many NNs would need to be considered before an intersection is found. Additionally, it is expected that if found, the interpolation will suffer from higher error due to use of facet vertices far from the interpolation point. Proper intersections of prediction points with the mesh are shown in blue.

triangulation must be recomputed, and if prediction coordinates change, the intersecting facets must be recomputed. Both triangulation and finding intersecting facets are computationally demanding with respect to memory and runtime for large datasets. For example, a mesh triangulation consisting of 50 000 points evaluated for 10 000 interpolation points requires ∼1.6 hours using 12 cores (∼20 CPU hours in total) with 128 GB of RAM available. The triangulation itself ($\sim 1 \times 10^8$ facets) requires ∼6 GB of memory storage.

5. Conclusion

We presented an application of barycentric interpolation on a 7-sphere which uses SVD transformations, linear projections, and NN searches to reduce the computational burden of high-dimensional hyperspherical triangulations and intersection-finding. Large point sets in high-dimensions still have large memory and runtime requirements, but are more tractable with these methods.

Acknowledgements

We thank Victoria Baird for useful discussions related to the visualization and triangulation of hyperspherical meshes. This work was supported by the National Science Foundation under Grant No. 1610077. This work was supported in part through computational resources provided by Brigham Young University's Office of Research Computing.

Glossary

NN nearest neighbor 1, 2, 5, 6

SVD singular value decomposition 1–4, 6

References

[1] T. Langer, A. Belyaev, H.-P. Seidel, Spherical barycentric coordinates, Proceedings of the fourth Eurographics symposium on Geometry processing (2006) 81–88. URL: http://portal.acm.org/citation.cfm?id=1281957.1281968.

[2] M. Floater, Generalized barycentric coordinates and applications, Acta Numerica 24 (2015) 161–214. doi:10.1017/S09624929. arXiv:1711.05337v1.

[3] M. Meyer, A. Barr, H. Lee, M. Desbrun, Generalized Barycentric Coordinates on Irregular Polygons, Journal of Graphics Tools 7 (2002) 13–22. doi:10.1080/10867651.2002.10487551.

[4] C. B. Barber, D. P. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Transactions on Mathematical Software 22 (1996) 469–483. doi:10.1145/235815.235821.

[5] D. Anisimov, C. Deng, K. Hormann, Subdividing barycentric coordinates, Computer Aided Geometric Design 43 (2016) 172–185. doi:10.1016/j.cagd.2016.02.005.

[6] M. Budninskiy, B. Liu, Y. Tong, M. Desbrun, Power coordinates: A geometric construction of barycentric coordinates on convex polytopes, ACM Transactions on Graphics 35 (2016). doi:10.1145/2980179.2982441.

[7] R. Dyer, G. Vegter, M. Wintraecken, Barycentric coordinate neighbourhoods in Riemannian manifolds, arXiv (2016). URL: http://arxiv.org/abs/1606.01585. arXiv:1606.01585.

[8] M. S. Floater, J. Kosinka, On the injectivity of Wachspress and mean value mappings between convex polygons, Advances in Computational Mathematics 32 (2010) 163–174. doi:10.1007/s10444-008-9098-z.

[9] K. Hormann, J. Kosinka, Discretizing Wachspress kernels is safe, Computer Aided Geometric Design 52-53 (2017) 126–134. doi:10.1016/j.cagd.2017.02.015.

[10] K. Hormann, N. Sukumar, Maximum entropy coordinates for arbitrary polytopes, Eurographics Symposium on Geometry Processing 27 (2008) 1513–1520.

[11] T. Langer, H. P. Seidel, Higher order barycentric coordinates, Computer Graphics Forum 27 (2008) 459–466. doi:10.1111/j.1467-8659.2008.01143.x.

[12] K. Lei, D. Qi, X. Tian, A New Coordinate System for Constructing Spherical Grid Systems, Applied Sciences 10 (2020) 655. doi:10.3390/app10020655.

[13] P. S. Peixoto, S. R. Barros, On vector field reconstructions for semi-Lagrangian transport methods on geodesic staggered grids, Journal of Computational Physics 273 (2014) 185–211. doi:10.1016/j.jcp.2014.04.043.

[14] P. Pihajoki, M. Mannerkoski, P. H. Johansson, Barycentric interpolation on Riemannian and semi-Riemannian spaces, Monthly Notices of the Royal Astronomical Society 489 (2019) 4161–4169. doi:10.1093/mnras/stz2447. arXiv:1907.09487.

[15] R. M. Rustamov, Barycentric coordinates on surfaces, Computer Graphics Forum 29 (2010) 1507–1516. doi:10.1111/j.1467-8659.2010.01759.x.

[16] V. Skala, Robust Barycentric Coordinates Computation of the Closest Point to a Hyperplane in En̂, Proceedings of the 2013 Internation Conference on Applies Mathematics and Computational Methods in Engineering (2013) 239–244.

[17] J. Tao, B. Deng, J. Zhang, A fast numerical solver for local barycentric coordinates, Computer Aided Geometric Design 70 (2019) 46–58. doi:10.1016/j.cagd.2019.04.006.

[18] J. Warren, S. Schaefer, A. N. Hirani, M. Desbrun, Barycentric coordinates for convex sets, Advances in Computational Mathematics 27 (2007) 319–338. doi:10.1007/s10444-005-9008-6.

[19] T. Francis, I. Chesser, S. Singh, E. A. Holm, M. De Graef, A geodesic octonion metric for grain boundaries, Acta Materialia 166 (2019) 135–147. doi:10.1016/j.actamat.2018.12.034.

[20] A. Morawiec, On distances between grain interfaces in macroscopic parameter space, Acta Materialia 181 (2019) 399–407. doi:10.1016/j.actamat.2019.09.032.

[21] R. Connor, L. Vadicamo, F. Rabitti, High-dimensional simplexes for supermetric search, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10609 LNCS (2017) 96–109. doi:10.1007/978-3-319-68474-1_7. arXiv:1707.08370.

[22] J. D. Boissonnat, R. Dyer, A. Ghosh, S. Y. Oudot, Only distances are required to reconstruct submanifolds, Computational Geometry: Theory and Applications 66 (2017) 32–67. doi:10.1016/j.comgeo.2017.08.001. arXiv:1410.7012.

[23] T. Anatoliy, Check if ray intersects internals of D-facet, Mathematics Stack Exchange, 2015. URL: https://math.stackexchange.com/q/1256236.