Spherical Models

Buttu D., Fialà N., Salicandro M.

Politecnico di Torino

June 17, 2025

An overview of the tasks

This project aims to accomplish the following tasks:

1. Creating a data structure whose properties of a polyhedron are saved in;
2. Given a set of four `valid` numbers (p, q, b, c), the software saves the properties of the polyhedron in the data structure at the first point. All polyhedrons will be inscribed in a sphere of radius 1.
3. Given a set of six `valid` numbers (p, q, b, c, id_D, id_A) where $id_D \neq id_A$ and $0 \leq id_D, id_A \leq |V| - 1$, our software computes the shortest path between the vertices id_D, id_A.

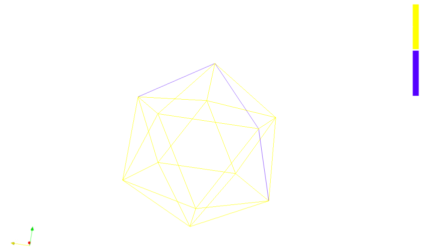


Figure 1: Example of short path (3)

Our code (UML documentation)

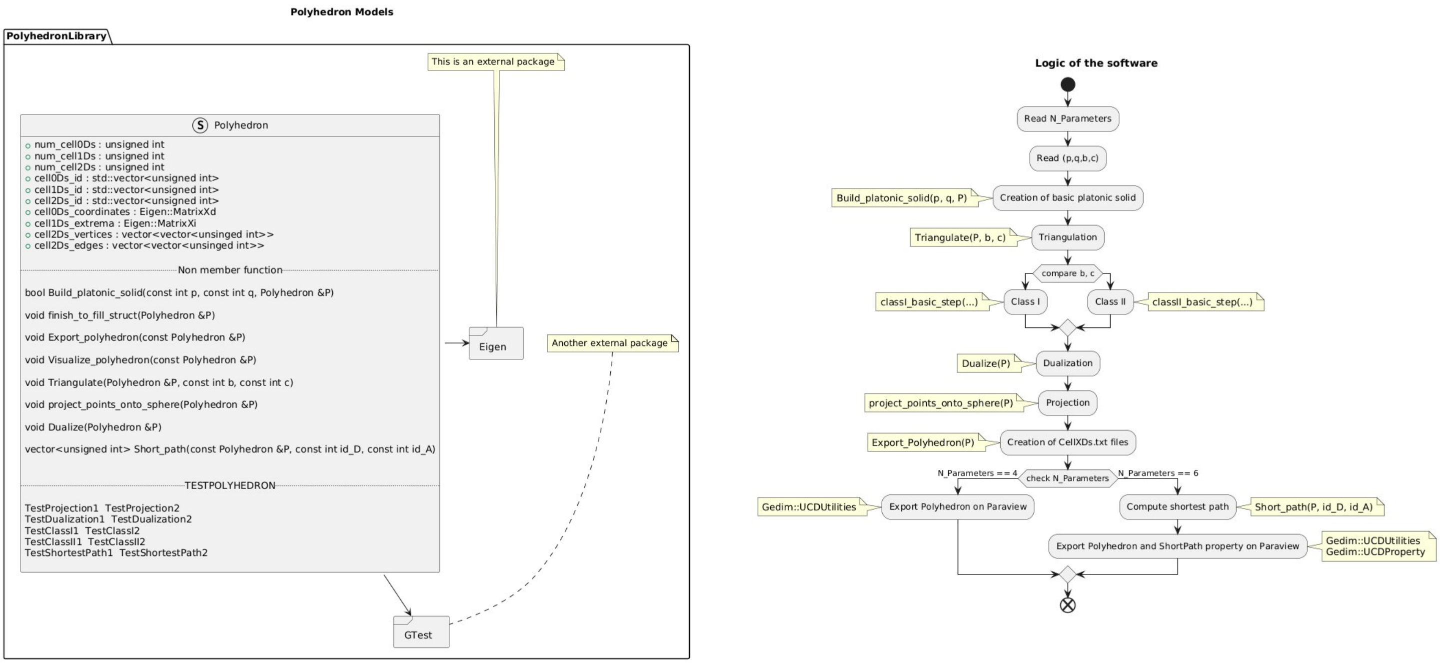


Figure 2: See more on our git repository

Polyhedron Structure

```
struct Polyhedron {
    // Element counts
    unsigned int num_cell0Ds;
    unsigned int num_cell1Ds;
    unsigned int num_cell2Ds;
    // Identifier vectors
    vector<unsigned int> cell0Ds_id;
    vector<unsigned int> cell1Ds_id;
    vector<unsigned int> cell2Ds_id;
    // Geometry storage
    MatrixXd cell0Ds_coordinates; // 3×N matrix
    MatrixXi cell1Ds_extrema; // 2×M edge endpoints
    // Topological relations
    vector<vector<unsigned int>> cell2Ds_vertices; // Faces vertex indices
    vector<vector <unsigned int >> cell2Ds_edges; // Faces edges indices
};
```

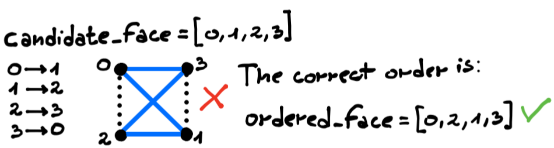
Dualize function

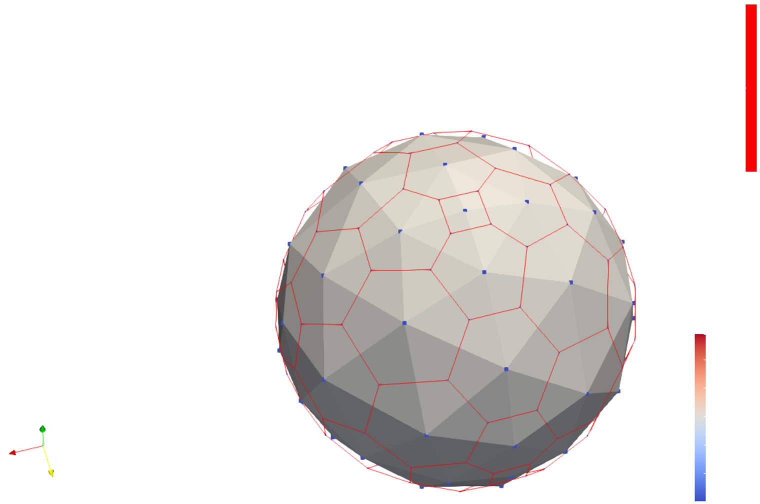
$O(F + Vk \log k)$

To dualize a polyhedron, we have to switch the roles of vertices and faces:

- $faces_{old} \longrightarrow vertices_{new}$
- $vertices_{old} \longrightarrow faces_{new}$

We can easily have the set of new vertices that concur to build a new face in the dualized polyhedron, but we don't have guarantee about their order.

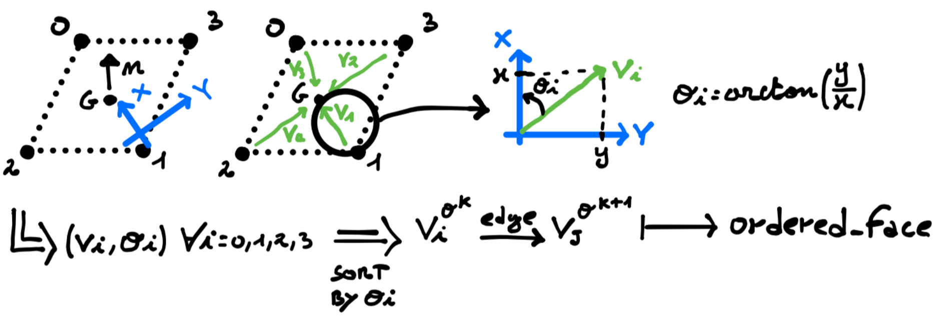




cycled_face_for_dual(...) $O(n \log n)$

It's known who are the new vertices of a face but we don't know how they are joined. Let's call this set S. We'll follow the following strategy:

1. We compute the normal verson $\vec{n}$ to the plane containing all elements of S (the plane is unique!)
2. We compute a local $\mathbb{R}^2$ system on the face;
3. Then for each $0 \leq i \leq |S| - 1$ we compute the angles formed by the $x-$axis of our new system and the vector $\vec{v} = \vec{G} - \vec{v_i}$.
4. Then the angles are sorted and we have a sort of a *clock* where we know exactly how vertices are connected.



Triangulate function (our code's ❤) $O(Fb^2V_{new} + V_{new}^2)$

Purpose: subdivide each triangular face of a polyhedron into smaller triangles using geodesic subdivision rules given by the math literature.

Four fundamental steps:

- **Initialize:** Prepare containers for new faces and a global vertex-ID map;
- **Process each face:**
  - retrieve vertices (A,B,C) of the current face,
  - subdivide into two cases :
    1. use classI_basic_step if b or c is non-zero ($\rightarrow$ class I),
    2. use classII_basic_step if b=c ($\rightarrow$ class II),
  - assign unique IDs to new vertices, avoiding duplicates
- **Reindex Triangles:** replace local vertex indices in new triangles with global IDs from the map,
- **Update polyhedron data:** compute new data using Euler's formula and finish the topology of the polyhedron

classl_basic_step(...)

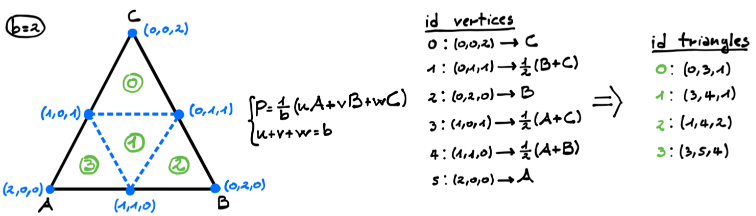
$O(b^2)$

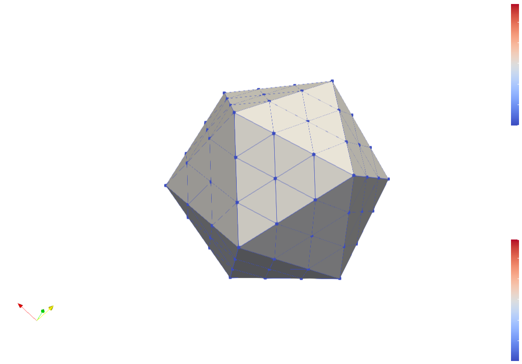
Let's fix a system of **barycentric coordinates**. Each point is of the form:

$$P = \frac{1}{b} \cdot (u \cdot A + v \cdot B + w \cdot C)$$

where u, v, w are **normalized**. Fixed u, v, then w is unique.
Given a line parallel to BC, then in the next vertex:

- v does not change;
- u increases of $1/b$.



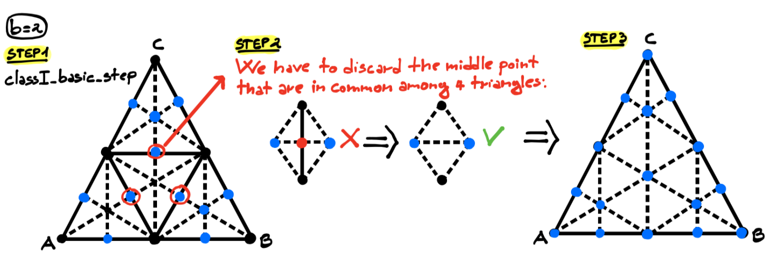


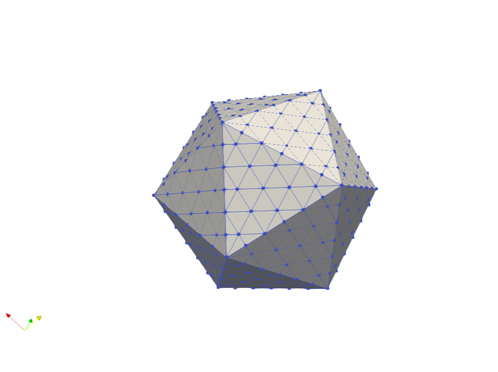
classII_basic_step(...)

$O(b^2)$

Let's take the points defined as in the previous page. In this way it is formed a uniform grid over the triangle.
Then it takes place a dual refinement :

- apply Class I subdivision $\rightarrow$ b^2 small triangles
- per each triangle:
  1. insert edge midpoints $M_{ij} = \dfrac{V_i + V_j}{2}$
  2. insert centroid $G = \dfrac{V_1 + V_2 + V_3}{3}$
  3. split into 6 triangles via G





Spherical projection

$O(V)$

This function centers the polyhedron at the origin and projects vertices onto a unit sphere, preserving structure while normalizing scale.

Original Mesh $\Rightarrow$ Centered $\Rightarrow$ Projected

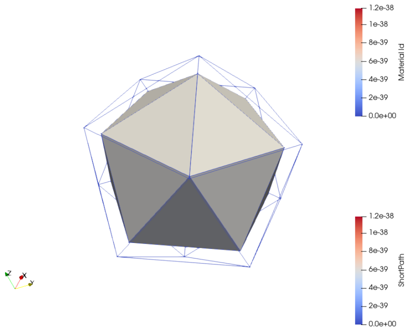


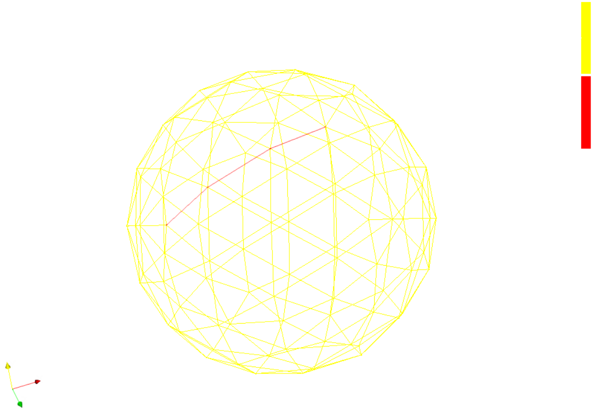
Figure 6: Projection of the icosahedron on the sphere

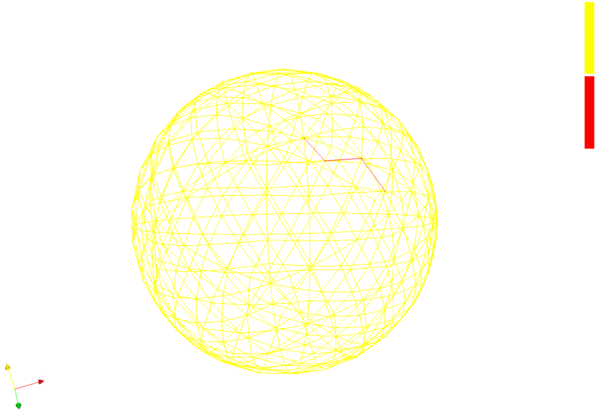
Short path

$O(V + E)$

Assumption: we consider the polyhedron as an unweighted graph, in this way we can calculate the shortest path in terms of edges crossed.

The main used tool is the BFS(Breadth-First Search) an algorithm that permits us with a few changes to easily find the path, rebuild it, and compute the required output.





Testing

1. **Spherical Projection** (2 tests)
    - Verify correct projection onto unit sphere
    - Test both normalized and non-normalized inputs
2. **Dualization** (2 tests)
    - Validate dual transformations
    - Tetrahedron and Octahedron cases
3. **Triangulation** (4 tests)
    - Class I (b=1,c=0) & (b=0,c=1)
    - Class II (b=c=1) & (b=c=2)
    - Verify vertex/edge counts and positions
4. **Shortest Path** (2 tests)
    - Validate geodesic path lengths
    - Check distance calculations on sphere

π approximation

Let's consider a Class I Polygon. When b is large enough, the polygon "tends" to a perfect sphere of radius 1. So we expect that the shortest path between the *North Pole* and *South Pole* tends to π. Indeed, that is!

```
./PCSProject 3 5 50 0 12963 7048
The shortest path that links 12963 and 7048 is 125 sides long
The shortest path that links 12963 and 7048 is 3.13671 long
```