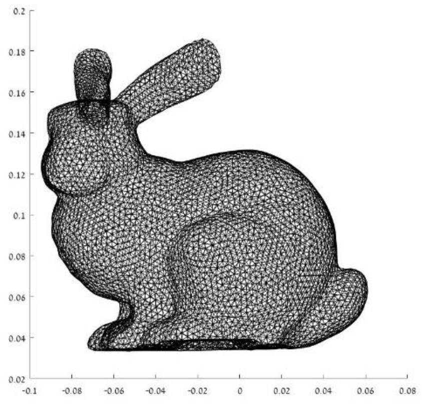
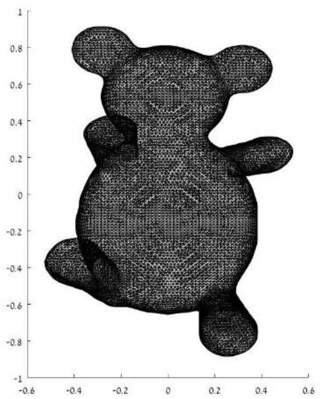
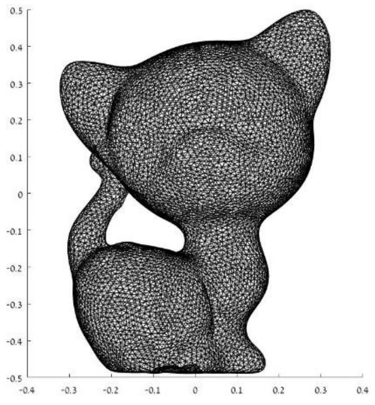
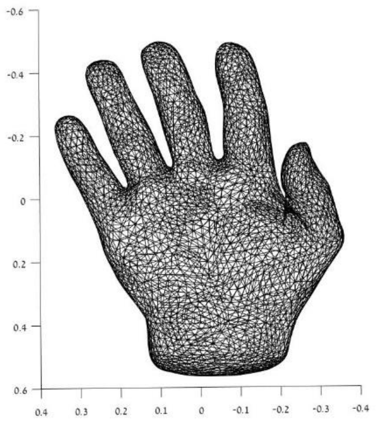
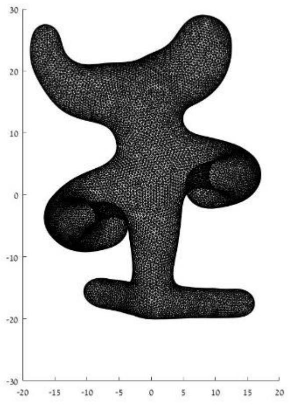
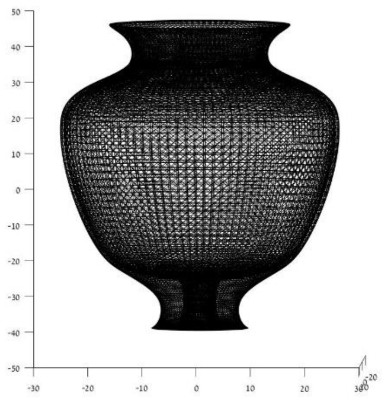
Digital Geometry Processing - 236329
Homework 2

316948694 201631349

1. Implemented in the functions – read_off.m and write_off.m.
   The number of vertices and faces that are specified at the beginning of the file were used to read those sections as chunks, without using loops.

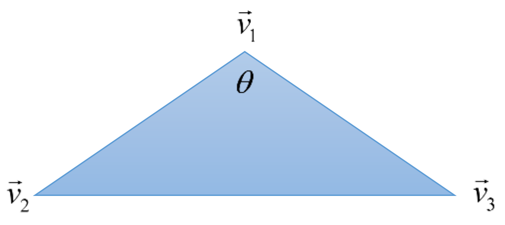
2. Implemented in class 'mesh'. The class consists of 4 fields:
   vertices – [V x 3] matrix containing vertices xyz coordinates.
   faces – [F x 3] matrix containing vertices indices.
   v_adj – [V x V] adjacency matrix describing adjacent vertices.
   vf_adj – [V x F] adjacency matrix describing adjacent vertices and faces.
   We used one of the suggested implementations from the tutorial for adjacency matrices.

3. Implemented in method 'plot_mesh' of class 'mesh'.
   Used 'patch' to plot the meshes.
   Here are several examples:

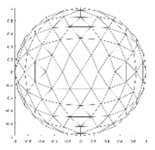







4. Implemented in methods 'faces_area' and 'vertices_area' of class 'mesh'.
   The area of each triangle was computed using the following formula:

$$\vec{u} = \vec{v}_1 - \vec{v}_2$$
$$\vec{w} = \vec{v}_1 - \vec{v}_3$$
$$\cos(\theta) = \frac{\vec{u} \cdot \vec{w}}{\|\vec{u}\|\|\vec{w}\|}$$
$$A = \frac{1}{2}\|\vec{u}\|\|\vec{w}\|\sin(\theta)$$

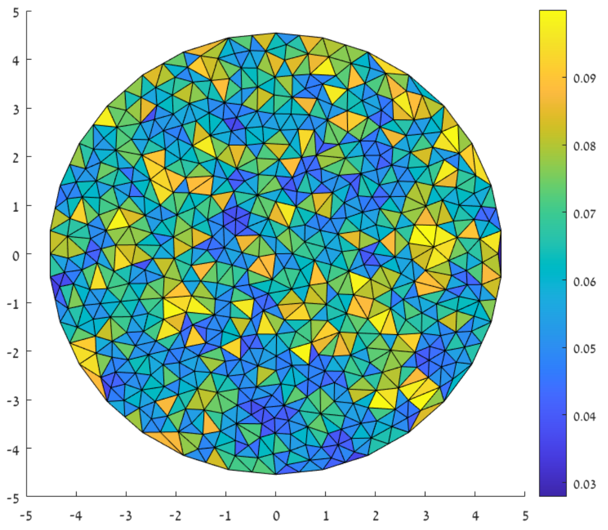


   The vertex area was computed using the face-vertex adjacency matrix. We modified the matrix to contain the area of each face instead of '1's by multiplication of each row in this matrix by a vector containing faces areas. Later we summed each row of the matrix and divided the sum by 3 to obtain vertices area.
   The division of each vertex area by 3 ensures that the sum of all face areas will be equal to the sum of all vertex areas, and that will be exactly the surface area. Since each face adjacent to 3 vertices, its area should be divided by 3, so its 3 neighbors will get an equal portion of its area.
   For example, for sphere_s0.off –
   the vertex and face areas are both equal to ~12.32.

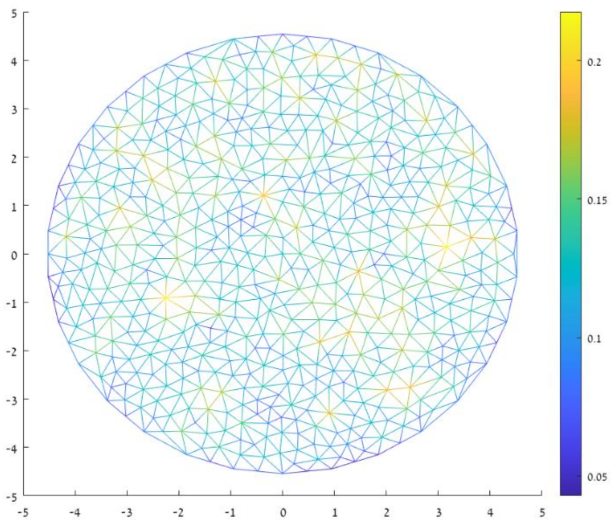


5. Implemented in method 'visualize' of class 'mesh'.
   The following examples visualize face and vertex areas:

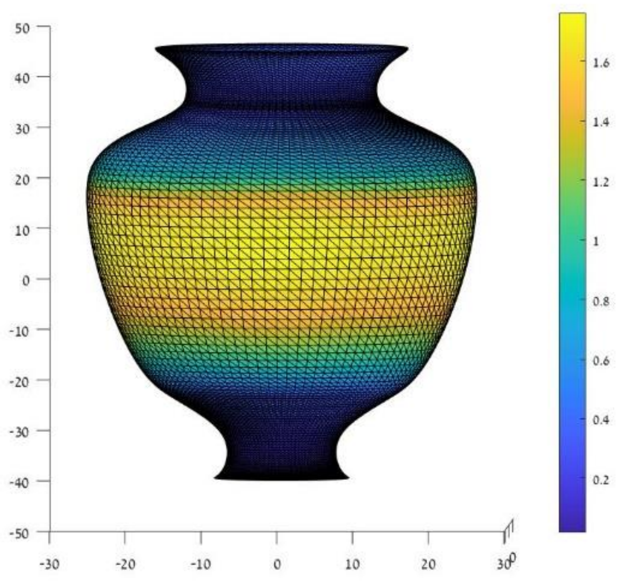
Face area



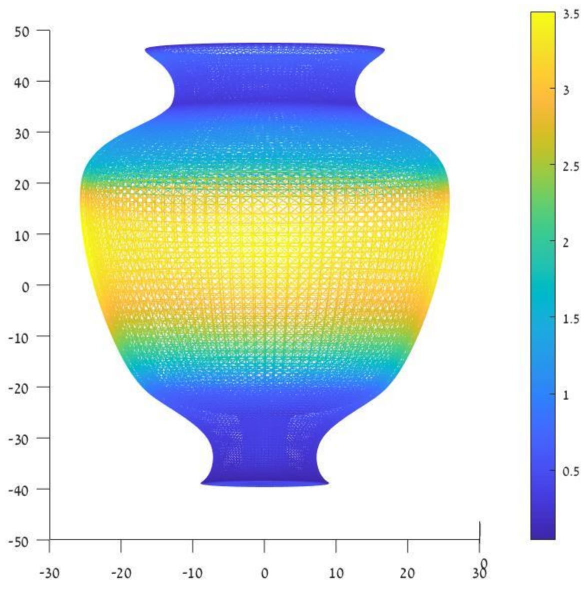
Vertex area



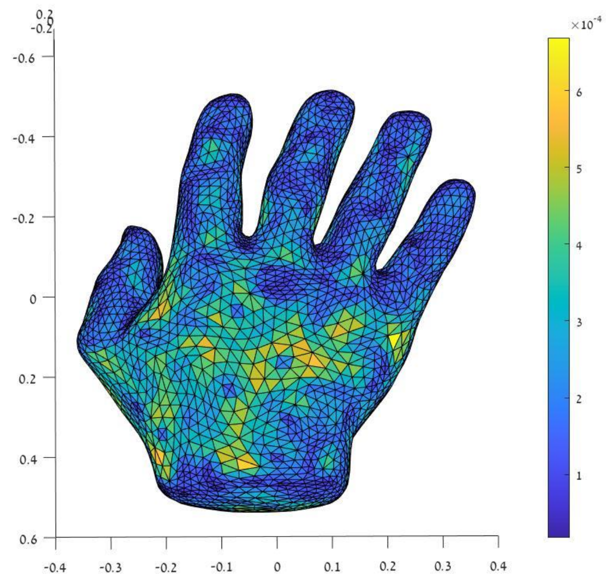
Face area



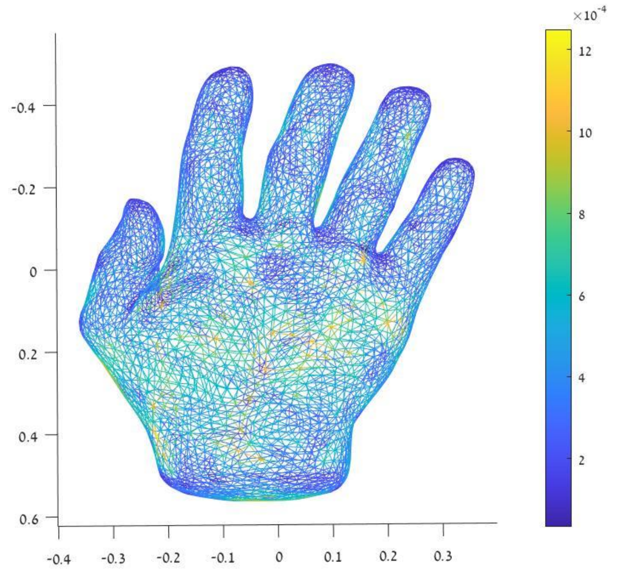
Vertex area



Face area



Vertex area



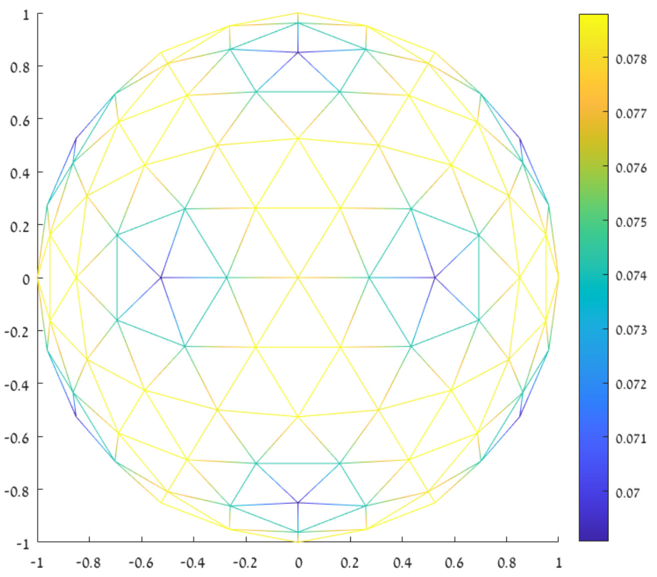
- Vertex areas are represented as edge colors. An edge that connects a vertex with a low area to a vertex with high area becomes brighter gradually.
- The disk contains relatively large triangles. It can be clearly seen how larger triangles get brighter colors.
- The two other visualizations (vase and hands) show how the relatively plain regions are represented with bigger faces, while the regions with high curvature represented with small triangles.

6. Implemented in methods 'interp_face_to_vertex' and 'interp_vertex_to_face' of class 'mesh'.
   Interpolation from faces to vertices, I_v^F [VxF], was done by dividing each element of the face-areas matrix, by each element of the vertex-areas matrix. The resulted matrix was multiplied elementwise by the vertex-face adjacency matrix to set to 0 the elements which are not adjacent to the relevant vertex as described in the specification.
   Calculation of I_F^V [FxV] - multiplying a matrix I_v^F by a diagonal matrix ***from left*** $A_F^{-1} I_v^F$ is equivalent to multiplication of each element in the diagonal of A_F^{-1} by ***each row*** of I_v^F.
   Multiplication of a matrix $A_F^{-1} I_v^F$ by a diagonal matrix ***from right*** $A_F^{-1} I_v^F A_V$ is equivalent to multiplication of each elements in the diagonal of A_V by ***each column*** of $A_F^{-1} I_v^F$.
   Regarding the back-and-forth interpolation -
   Mathematically, the inverse for [mxn] matrix A with full rank rank(A) = min{m,n} will be as follows:
   - If m<n then A has a *right inverse* given by: $A_{right}^{-1} = A^T \left(AA^T \right)^{-1}$
   - If m>n then A has a *left inverse* given by: $A_{left}^{-1} = \left(A^T A \right)^{-1} A^T$

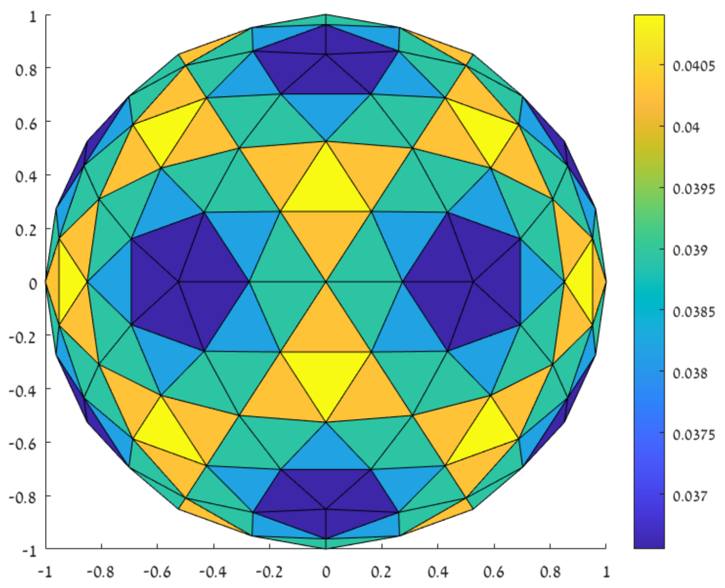
   The rank of I_v^F matrix is < min{m,n} (**adjacent faces** to vertex can be computed through **adjacent vertices**) hence it is not invertible. If the $\left(I_F^V \right)^T A_v$ matrix holds the second condition (V<F), it may be inverted with the specified *left inverse* matrix. The result of back-and-forth interpolation of the vertices is: $\tilde{v} = I_v^F A_F^{-1} \left(I_F^v \right)^T A_V v$. If in some cases the *left inverse* matrix is obtained in the last expression, it may be inverted.

   Examples of back-and-forth area interpolation in the next pages. One can notice that the areas of the vertices remain similar to the original, while faces areas are different.

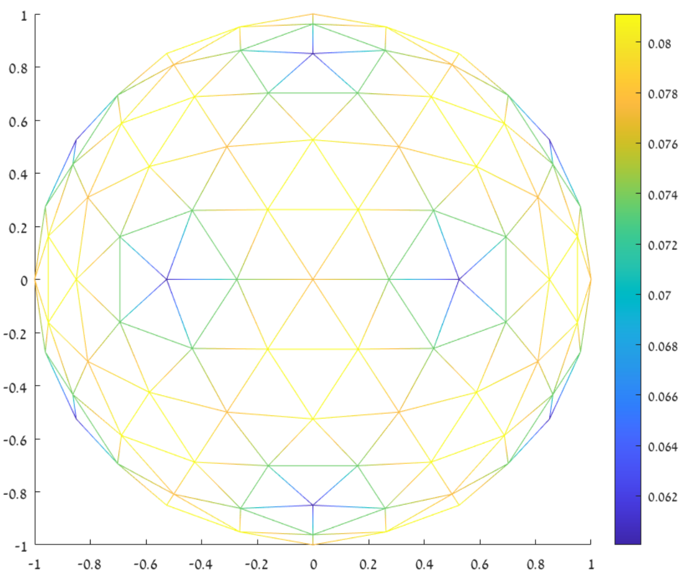
Vertex area - Interpolated



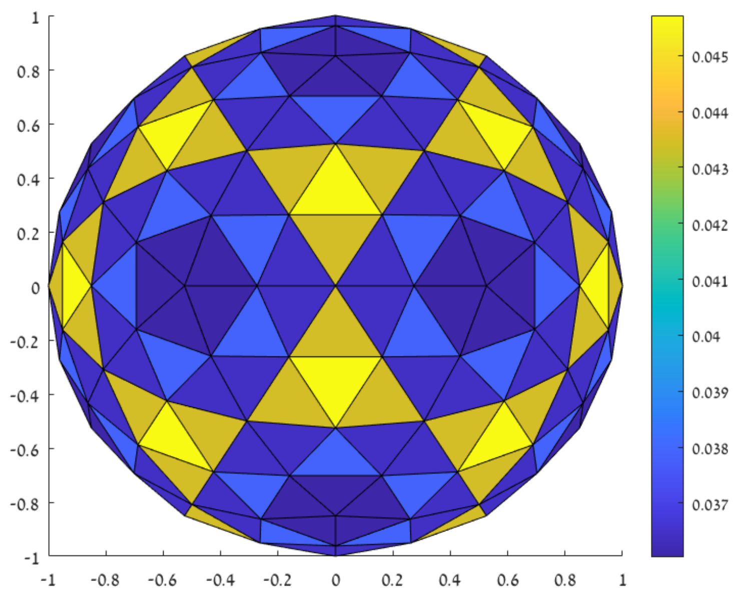
Face area – Interpolated



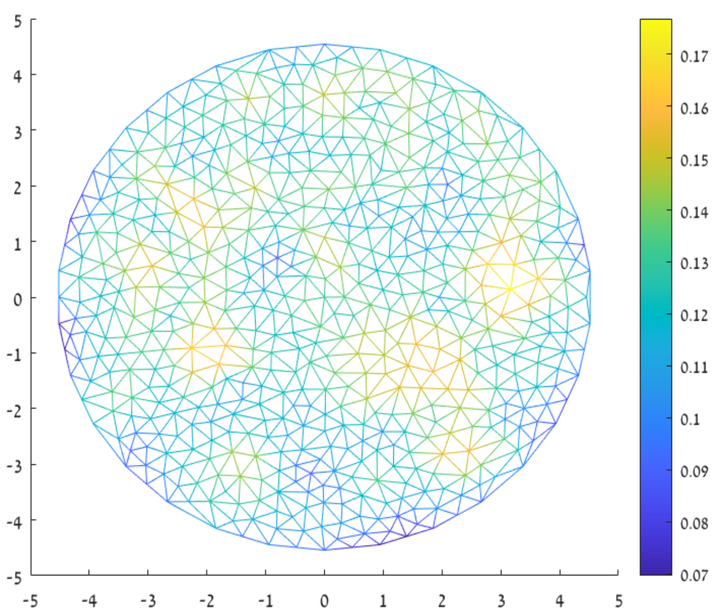
Vertex area - Original



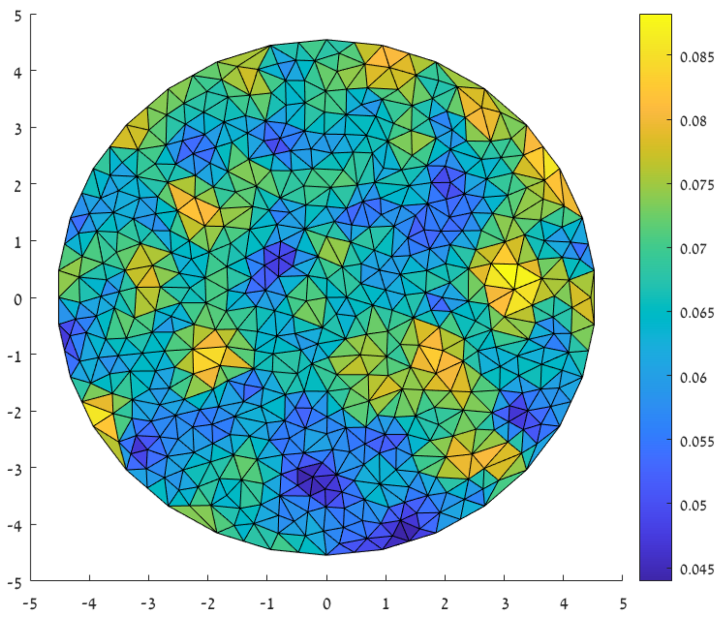
Face area - Original



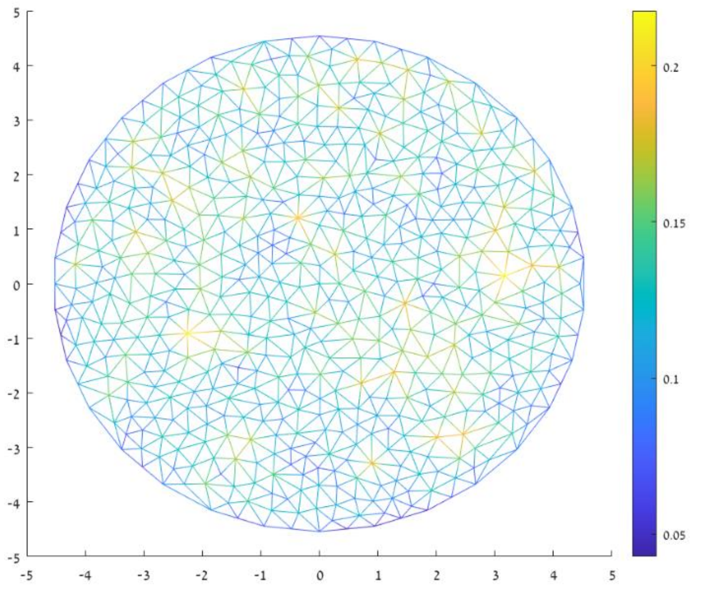
Vertex area - Interpolated



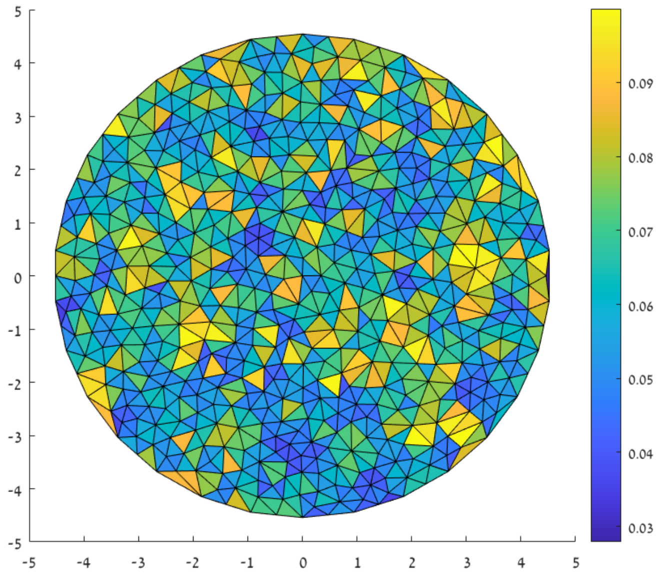
Face area - Interpolated



Vertex area - Original



Face area - Original

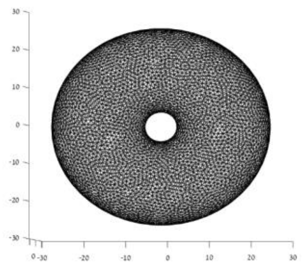


7. Implemented in methods 'get_boundary_edges' and 'get_genus' of class 'mesh'. We found boundary edges by finding all edges with one adjacent face. The genus was calculated using following formulas:

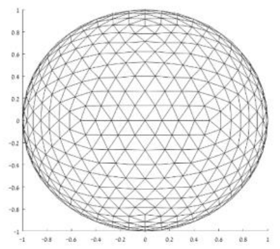
$$\chi = V - E + F$$

$$genus = \frac{1}{2}\left(2 - boundary\_elems - \chi\right)$$

The torus has genus = 1

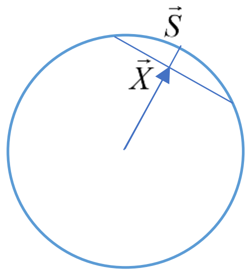


The sphere has genus = 0



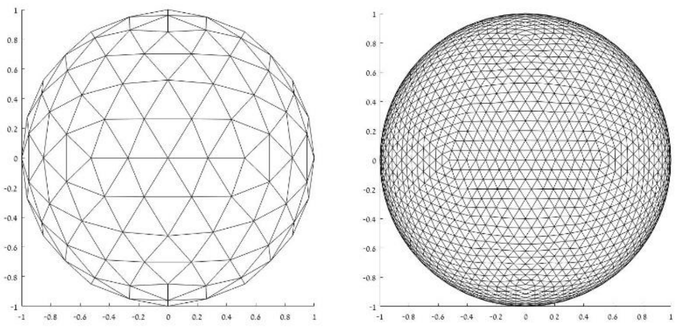
Analysis

1. Implemented in function 'tri_error_unit_sphere'.
   As all points $\vec{S}$ have a length of one unit, the vector $\vec{S}$ can be constructed by only computing $\vec{X}$ direction:

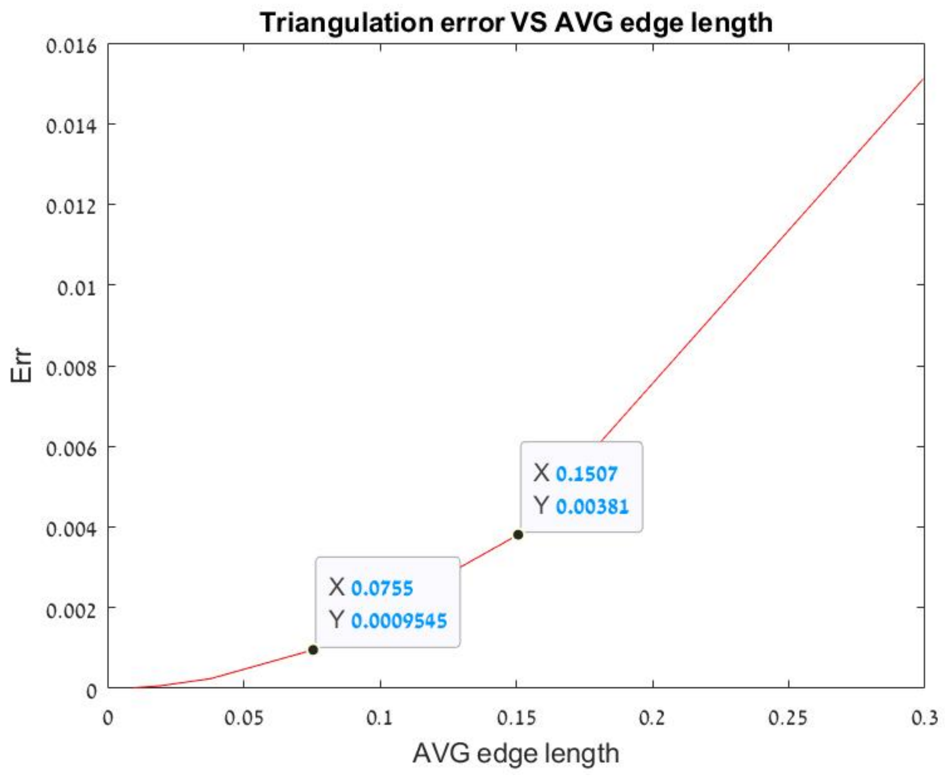


$$\vec{S} = \frac{\vec{X}}{\left\|\vec{X}\right\|}$$

   Examples of two unit spheres:



   Triangulation error plot:

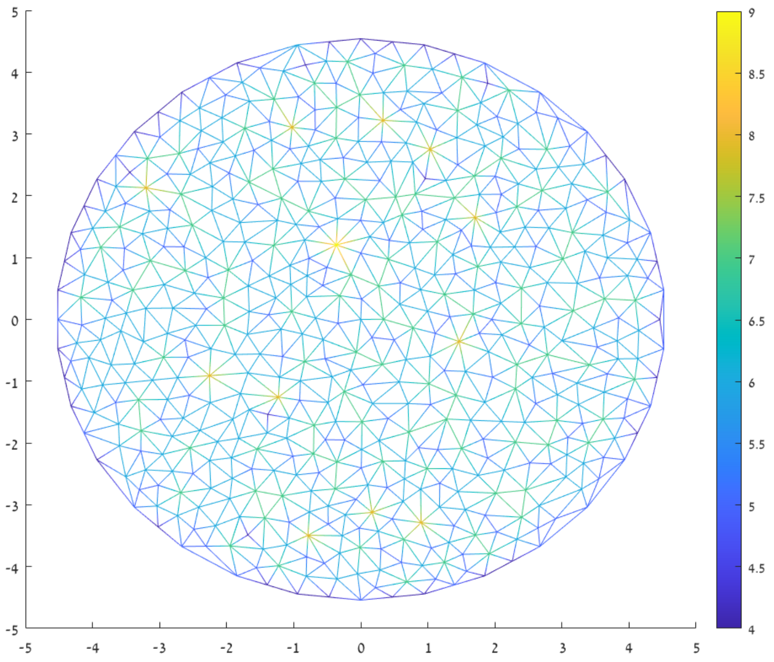


One can notice that the error is $O\left(n^2\right)$ where n is the average edge length, as we saw in class.

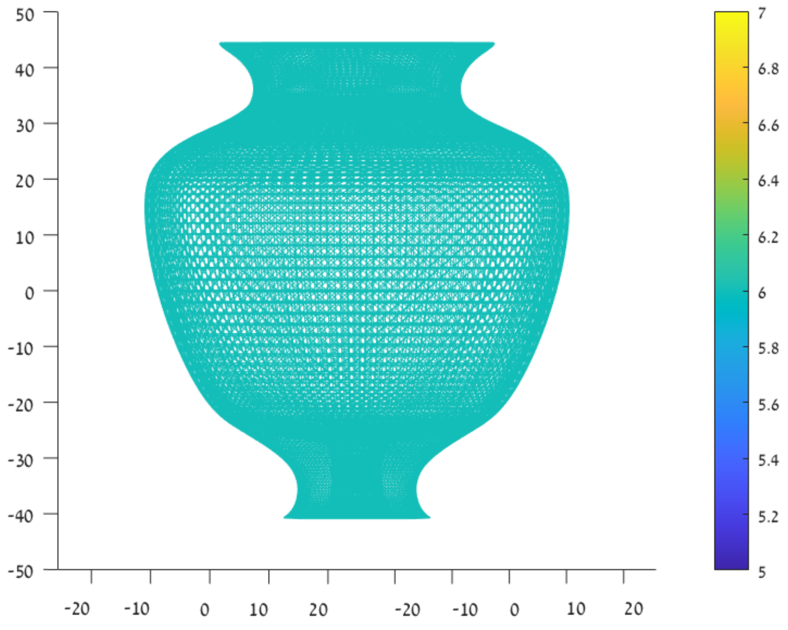
For example, when the average edge length is reduced by 2: $\frac{0.1507}{0.0755} \approx 2$

The error is reduced by 4: $\frac{0.00381}{0.0009545} \approx 4$

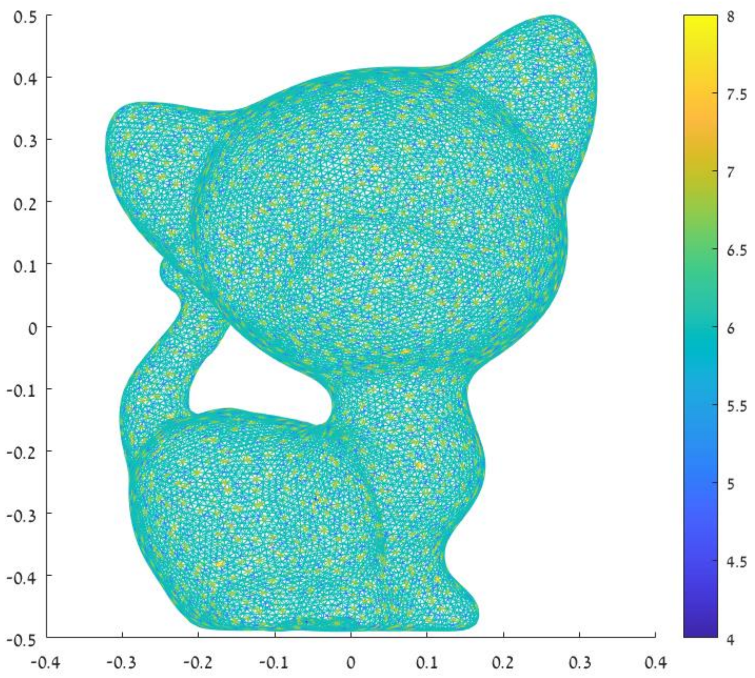
2. Implemented in method 'calc_valence' of class 'mesh'.
   The valence was calculated by directly using the vertex adjacency matrix.
   Some examples:



One can notice that boundary elements have relatively small valence values.
The average valence is 5.8.

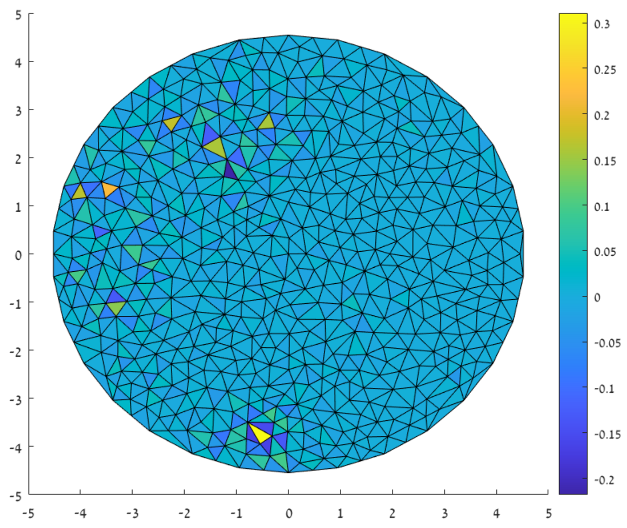


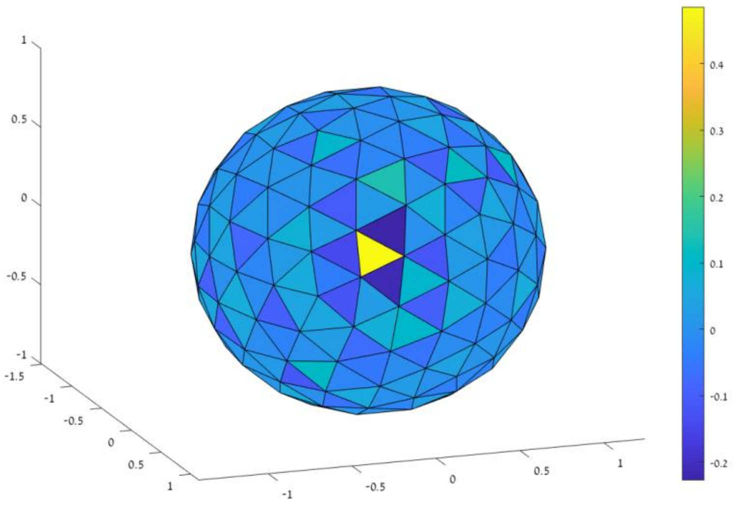
The vase has constant valence which is equal to 6.



Average valence is 6.

3. The rank of I_v^F matrix is < min{m,n} as was specified in a previous section, hence it doesn't have a kernel.
   Here are some examples of the kernel computed for I_F^v (it has a kernel and hence it is not invertible, there are vectors that are mapped to 0 and cannot be recovered):

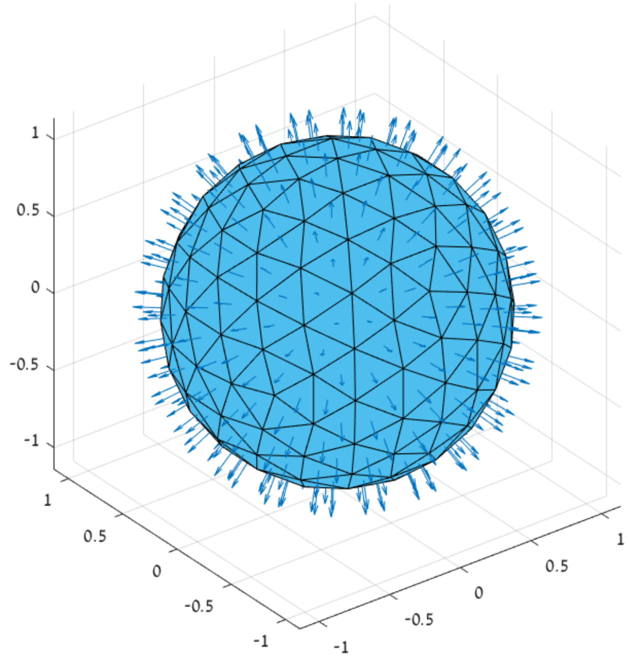


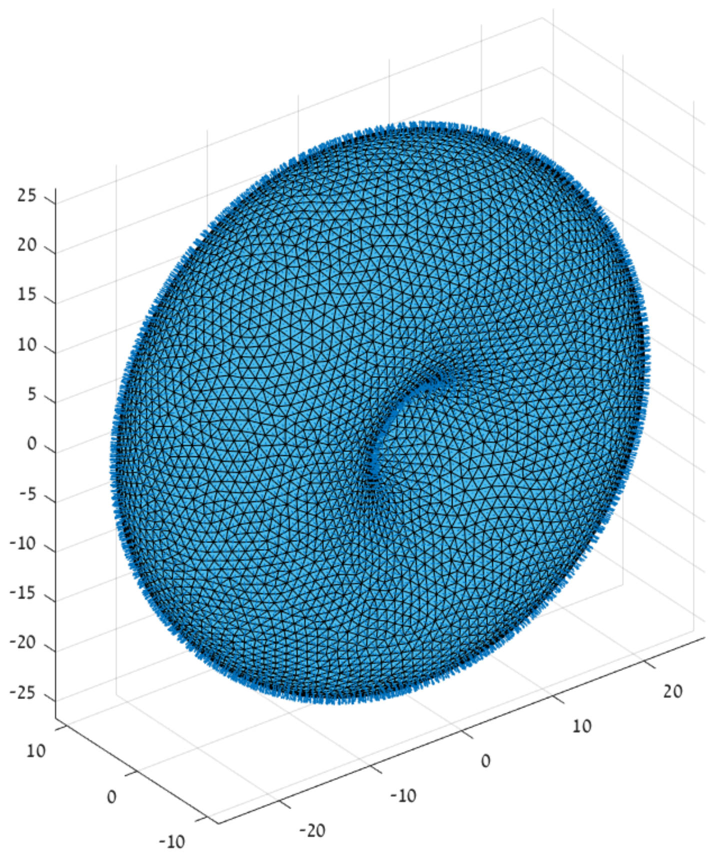


In both examples there are negative values.

Two additional tasks:

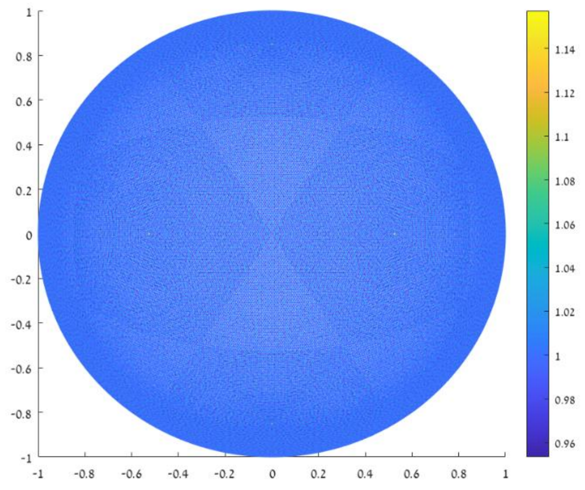
4. Normal calculation and visualization – implemented in method 'calc_normal' of class 'mesh'.
   The normal is calculated using cross product, and the visualization was done using quiver3.
   Examples



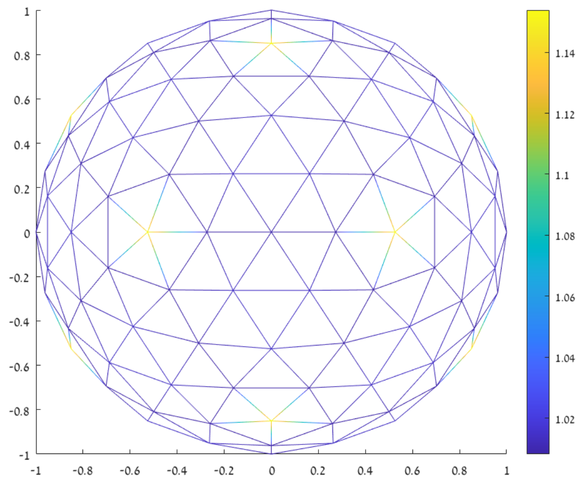


5. Gaussian curvature – implemented in method 'calc_gauss_curv' of class 'mesh'.
   The curvature was calculated using the formula $G = \frac{\left(2\pi - \sum_j \theta_j\right)}{A_v}$. To efficiently calculate this term, we modified the vertex-face adjacency matrix to contain the angles instead of '1's.

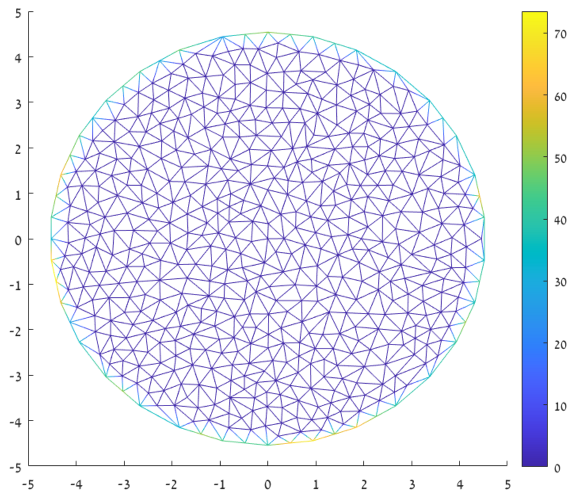
   Examples in the next page.



For a dense unit sphere, the gaussian curvature is approximately 1 at all vertices.



For a less dense unit sphere, whenever the valence is getting lower the curvature is getting higher.



Disk curvature is approximately constant 0 except of the boundary.