Proceedings of the IMProVe 2011
International conference on Innovative Methods in Product Design
June 15th – 17th, 2011, Venice, Italy

Development of an algorithm for a triangular approach to the sphere based on the Platonic solids using MATLAB scripts.

M. Heredia Conde [a], Manuel Pérez Vázquez [b], José M. Gomis Martí [c]

[a] Engineering Design Department collaborator (University of Vigo - Spain)
[b] Engineering Graphics an Design professor (University of Vigo - Spain)
[c] Enginering Graphics professor (Polytechnic University of Valencia - Spain)

Article Information

Keywords:
Sphere,
Triangular approach,
Platonic bodies,
MatLab.

Corresponding author:
Manuel Pérez Vázquez
Tel.: +34 677689233
Fax.:+34 986812201
e-mail: maperez@uvigo.es
36201 Vigo (Pontevedra) SPAIN

Abstract

The sphere is a common object in uncountable engineering problems, which not only appears in structural elements like domes but also in thousands of mechanisms normally used in diverse kinds of machines. To design, calculate and analyze the behaviour on service of spherical elements it's essential to have a good method to create an ordered group of discrete points of the spherical surface from the parametric equations commonly used to define the sphere continuously.

One of the best known and widely used in high-level programming environment is MATLAB. The programming language has thousands of functions, lots of them specially designed for engineering processes. One of these functions generates a sphere knowing a given radius and shows the result. Nevertheless, this function is really imprecise because it's based on parallels and meridians besides the obtained vertices don't keep a constant distance each other. That because it would be appropriate to design a new function to generate accurate discrete approximations of the sphere.

The objective of this paper is create a low-level function in MATLAB to obtain a discrete sphere with high regularity and high approximation in order to provide a good base to solve sphere-based engineering problems. To ensure a perfect symmetry and a high regularity platonic bodies will be used as a base to divide the continuous spherical surface in a finite number of regular triangles. The obtained results for the different seed bodies will be represented graphically and compared to each other. The accuracy of each method will be evaluated and compared too.

1 Introduction

The objective of this work is to design a program that allows triangulation of a sphere defined by the value of its radius (eq. 1, eq. 2, eq. 3; being R the radius of the sphere, θ the zenithal coordinate and ϕ the azimuthal coordinate) [1], [2]. The process of triangulation [3], [4] involves the calculation of new vertices of the polyhedron which will replace the given sphere, its proper association forming triangles, which are the faces of the polyhedron, and finally, the graphic representation in a three-dimensional environment of those faces, giving rise the polyhedron result of triangulation, whose vertices have to meet the necessary condition of being contained in the initial sphere. According to these last conditions it follows that for the fulfilment of these objectives the program will work generating polyhedra inscribed in the given sphere, discarding the circumscribed polyhedral approach option.

$$X = R \cdot \sin\theta \cdot \cos\phi \qquad (1)$$

$$Y = R \cdot \sin\theta \cdot \sin\phi \qquad (2)$$

$$Z = R \cdot \cos\theta \qquad (3)$$

1.1 State of art

MATLAB provides developers a huge amount of generic functions, many of them oriented to create and show three-dimensional surfaces. Between these functions there is one that allows us to generate a three-dimensional mesh for the surface of a sphere: "SPHERE(N)". This function creates a mesh of the spherical surface, where the radius of the sphere is one unit and "N" is the number of divisions in which the vertical diameter of the sphere is divided by the parallels planes. So "N" is the number of meridians, or the number of parallels plus one, used to create the mesh from the theoretical sphere.

At first one could think that this approach is enough to cover any kind of need for obtaining a discrete sphere. The lack of accuracy derived from a low value of the parameter "N" could be easily solved increasing its value and, therefore, the computing time. Nevertheless, a deeper analysis would show that this approach preserves a sensible difference between the longest and the shortest edges of the mesh, even increasing the value of the parameter "N". This detail could be not so important if the approach is only used to obtain a three-dimensional rendering with no further intention but it becomes critical when the obtained data (points and edges of the result mesh) are used in engineering calculus like, for example, resistance of structures.

In order to compare the results of the developed algorithm with the results offered by the MATLAB built-in spherical approach the divergence between the shortest and the longest edge of the approach will be calculated for several values of the parameter "N". This divergence will be used as a measure of the regularity of the approach.

The first step to calculate the shortest and the longest edges of the mesh is to determine the radius of the smallest and the biggest parallels (excluding equator) of the geographic coordinate system (meridians and parallels) used to divide the sphere. The radius of the smallest parallels, which are the closest to the poles is given by the eq. 4, where "R" is the radius of the sphere and "n" is the number of vertical divisions made by the parallels.

$$r = \sqrt{1 - \frac{(n-2)^2}{n^2}} R \qquad (4)$$

The radius of the biggest parallels, which are the closest to the equator, are given by the eq. 5 (if "n" is odd) and the eq. 6 (if "n" is even), where "R" is the radius of the sphere and "n" is the number of vertical divisions made by the parallels.

$$r = \sqrt{1 - \frac{1}{n^2}} R \qquad (5)$$

$$r = \sqrt{1 - \frac{2^2}{n^2}} R \qquad (6)$$

The longest edge between the edges that form the meridians is extracted from the radius of the smallest parallels and its value is given by the eq.7.

$$a = \frac{2R}{\sqrt{n}} \qquad (7)$$

The shortest edge between the edges of the meridians is extracted from the radius of the biggest parallels and its value is given by the eq.8 and the eq. 9, depending on if "n" is odd or even, respectively

$$a = \frac{2R}{n} \qquad (8)$$

$$a = \sqrt{2\left(1 - \sqrt{1 - \frac{2^2}{n^2}}\right)} R \qquad (9)$$

The parallels edges are obviously calculated from the radius of the corresponding parallels: the longest edge is obtained from the biggest parallel radius and the shortest edge is calculated from the smallest parallel radius, both of them following the eq. 10, where r is the corresponding radius of the considered parallel.

$$a = 2 \cdot r \cdot \sin\left(\frac{\pi}{n}\right) \qquad (10)$$

Using the obtained equations a complete study of the edges divergence of the geographic coordinate system-based MATLAB own spherical approach algorithm is carried on for several values of the "N" parameter, in order to obtain solid data based on a commercial solution to compare our algorithm results with.

The achieved results are shown in the table Tab. 1, where the first row shows the divergence between edges of the parallels, the second row shows the divergence between edges of the meridians and the last row contains the global divergence of the edges of the approach. Each column corresponds to a different simulation using a different value of "N" and, consequently, a different number of generated faces (N^2).

Faces (n^2)	Parallels	Meridians	Global
5^2	0.2114	0.4944	0.7518
10^2	0.2472	0.4314	0.4314
15^2	0.2075	0.3831	0.3831
20^2	0.1764	0.3470	0.3471
25^2	0.1522	0.3200	0.3200
30^2	0.1340	0.2984	0.2984
35^2	0.1195	0.2809	0.2809
40^2	0.1079	0.2662	0.2672
45^2	0.0983	0.2537	0.2570
50^2	0.0904	0.2428	0.2477

Tab. 1 Divergence between edges

The table shows results of the spherical approach regularity (divergence between edges) analysis applied over the MATLAB "SPHERE" algorithm. The values are given as multiples of the radius of the original sphere (R).

The evolution of the global edge divergence obtained using the MATLAB "SPHERE" algorithm increasing the value of "N" is our estimator of the regularity of the approach and is graphically shown in the fig. 1. The divergence (blue line) is the difference between the longest edge (red line) and the shortest one (green line).

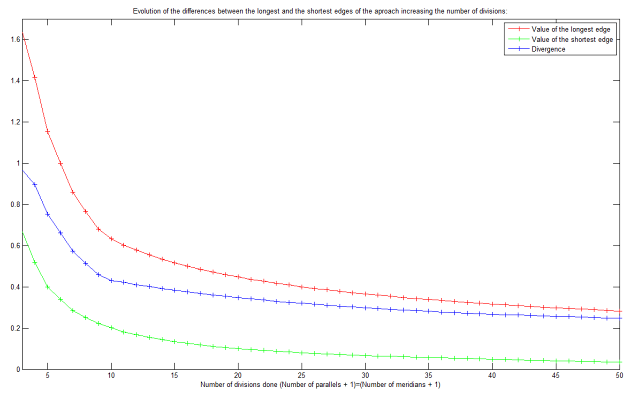

Fig. 1 Evolution of the divergence between edges by increasing the value of the parameter "N".

2 Development of the program

2.1 Main structure and basic lines of the program

In order to achieve the proposed objectives multiple algorithms [5], [6], [7] can be designed for generating polyhedral surfaces, starting from different basic polyhedra. However, if you want to make a high quality triangulation, namely ensuring that the resulting polyhedron vertices are evenly distributed across the surface of the sphere and that the faces of the generated polyhedron are identical, it is clear that you have to start to work over a regular polyhedron whose faces are triangles. Between all the Platonic solids there are three that meet this condition: the tetrahedron, the octahedron and the icosahedron, with four, eight and twenty equilateral triangles respectively. All of them are therefore valid seed polyhedra to initiate a precise triangulation of the sphere. Despite not being made up of triangles, the cube deserves special attention, since it is readily apparent that if we draw the two diagonals of each of their faces we get four new triangular faces from each of the initial square faces. However, it is also seen that to create a valid and operative seed polyhedron from this cubic approach it is necessary to move the new vertices (centres of cube faces) along the perpendicular line to the face that cut the face of the cube in its centre, until they meet the cube circumscribed sphere membership condition. Once we have modified the cube we obtain a twenty-four non-equilateral triangular faces polyhedron which is capable of being used as a basis for the triangulation of the sphere.

Having analyzed the alternatives, we decided to use the four polyhedra mentioned above to develop four different models of triangulation of the sphere. Three of them will generate a net of equilateral triangles: The tetrahedral triangulation [8], the octahedral triangulation and the icosahedral triangulation; and the other one will create a network of non-equilateral triangles derived from the cube polyhedron described above: Cubic triangulation.

Each triangulation method need, first at all, to generate the corresponding seed polyhedron, taking the radius of the sphere demanded by the user as the only input data. To generate the tetrahedral seed, inscribed into the theoretical sphere, it is necessary to use the eq. 4 to calculate the length of each edge of this regular polyhedron (a) from the radius of the sphere (R). The length of each edge of a cube inscribed into a sphere is given by the eq. 5 and it’s the first data used to create the cube-derived triangular-faced seed polyhedral. The octahedral seed can be generated like the tetrahedral one using the eq. 6 to obtain the proper edge length. Finally the icosahedral seed is generated in the same way using the eq. 7 to calculate the corresponding edge value.

$$a = \frac{4}{\sqrt{6}} \cdot R \qquad (4)$$

$$a = \frac{2}{\sqrt{3}} \cdot R \qquad (5)$$

$$a = \frac{2}{\sqrt{2}} \cdot R \qquad (6)$$

$$a = \frac{4}{\sqrt{10 + 2 \cdot \sqrt{5}}} \cdot R \qquad (7)$$

Once both the objectives and the *modus operandi* of the program are fully defined, it’s necessary to establish the form that it has to have, in terms of structuring and encapsulation of the code refers [9], [10], [11]. Since four tasks are performed in parallel on the same program, a base program and another four specific sub-programs will be used. The main program will interact with the user and will call one of the four sub-programs, which will perform the necessary calculations and display the final result obtained, as appropriate to the user's choice.

Once at this point we must define the type of program that will be obtained after the compilation of the code and the programming language that will be used to write the corresponding code. In order to allow the later use of the generated program as part of medium and high level engineering programs we decided to adopt as kind of program the ".m" function type, which corresponds to a MATLAB own kind of file. MATLAB is a worldwide well-known computer program for advanced calculation based on matrices [12], [13], [14], [15], [16], [17], [18], [19], [20], [21].

The wide use of MATLAB in almost all the branches of the applied sciences, and even more in the engineering world, will make possible to include our function in a huge range of applications. MATLAB also offer us the possibility of converting the program into an universal C library, which can be included in any other program. In order to program this type of executable MATLAB has its own language and a compiler called "Editor", from which programs can be written, apart from the command window itself, or "Command Window", which you can perform any operation or all of them directly, without creating a corresponding program in a ".m" file using the editor. In our case we use the command window only for testing or simple calculations may be necessary isolated in the process of project development and eventual debugging. The necessary software will be developed exclusively as ".m" files using the editor.

In the coming paragraphs we will analyze the base program and the various sub-programs that are invoked from it.

2.2 Main program

The base program has only one task: to determine the approximation of the sphere that the user (or, in its case, higher level program that calls the function) wants to perform and to execute the corresponding program for this approach. To obtain the user’s preferences (if the function is called from a higher level program the necessary information will be passed as parameters to the function), the program prints out a list of the four options available to the user and urges him to choose one providing the numeric value associated with it and corresponding to the order they are listed. Once stored this value in a variable it is contrasted with the values for each option to find the selected option. When the user’s choice is found, the main program calls the program that performs the chosen spherical approximation and the execution of the main program finishes at the end of the called subroutine. If the value entered by the user does not match with any of the options available, the program will repeat the question until the user gives a coherent response.

The base program name will be Sphere.m, so to start running our suite of triangular approximations of the sphere simply type "Sphere" in the MATLAB command window after making sure that we are working on the proper "Current Directory".

The sub used to calculate and graphically display the different approaches of the sphere will be detailed later and are the following ones:

- Triangulation_tet.m performs the tetrahedral triangulation of the sphere.
- Triangulation_cub.m performs the cubic triangulation of the sphere.
- Triangulation_oct.m performs the octahedral triangulation of the sphere.
- Triangulation_ico.m performs the icosahedral triangulation of the sphere.

2.3 Structure of each triangulation of the sphere algorithm

The first step when we start writing code for any of the sub files -Triangulation_tet.m, Triangulation_cub.m, Triangulation_oct.m, Triangulation_ico.m- should be to delete all the variables that will be used during the execution of the program, in order to avoid that previous values interfere giving inconsistent results. In our case, since the programs do not use any previous variable stored in the workspace, we decided to delete all of the variables using the command "clear." This step will always take place at the beginning of all the sub-programmes.

Then, it proceeds to ask the user (or higher level program) for the radius of the sphere whose triangulation approach he wants to perform and the number of divisions that will take place, after triangulation, in each of the edges of the corresponding seed polyhedron (tetrahedron, cube, octahedron or icosahedron) inscribed into the sphere which radius was introduced above. Note that this last value corresponds to the square root of the number of triangles in which each of the faces of the inscribed polyhedron is subdivided, that is to say, if you enter a value "n", n^2 triangles will be generated for every initial triangular face. This is a precision parameter of the approach to be made. A too high value of this parameter will provide more accurate results but will involve a greater number of points that may make it harder the visualization of the 3D generated polyhedron. The increase in computing time is not significant for reasonable values of precision. Anyway, the use of the function in real-time systems or any other kind of high level programs that require high speed execution would force us to establish a balance between the accuracy of our results and the meeting of the real-time requirements.

A POINTS matrix (TET_POINTS, CUB_POINTS, OCT_POINTS or ICO_POINTS, in each sub) is constructed with the vertices of the inscribed polyhedron sorted by rows and expressed in spherical coordinates, so that the first column contains the radial component, the second one the azimuthal component and the third one the zenithal component. These last two coordinates are given in radians, while the radius is expressed in the same units used by the user to enter the value.

Then the FACES matrix (TET_FACES, CUB_FACES, OCT_FACES or ICO_FACES) is generated by combining the rows of the corresponding POINTS matrix (vertices of the tetrahedron) in groups of three to form the different triangular faces (three points each) of the tetrahedron.

Once these matrices are generated, we construct the ORDER array, which consists of sets of three numbers that are the indices of three of the points that will be obtained later, when the division of each triangular face of the tetrahedron is performed and whose mission is to define in what order they join the new generated points to form the new faces. The indices are assigned to each face from top to bottom and from left to right with the top vertex (or bottom in some other cases) facing upward, as shown in the fig. 2.

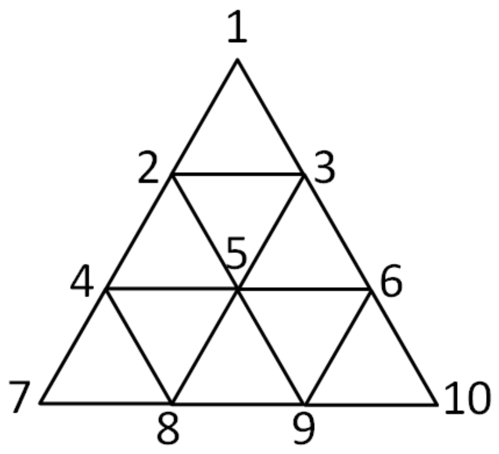

Fig. 2 Scheme of triangulation system

The figure also shows the scheme of the triangulation system carried on in order to obtain new equilateral triangles from each of the equilateral triangular initial faces of the corresponding seed polyhedron.

The cubic approach doesn't provide an equilateral-faced seed polyhedron so this triangulation system can't generate equilateral faces from the original ones, but it preserves the similarity between the children triangles and the mother one.

The numbers show the order followed to generate all the needed points from the spherical coordinates of the points of the original triangular face (1, 7, and 10).

According to the description of the matrix ORDER and looking at the Figure 2, the vectors (1, 2, 3) and (2, 3, 5) will be, for example, rows of ORDER. The size of this matrix will be n^2x3, since each one of the n^2 generated faces per original face has three points. Now that it is has been set the order in which we must connect the dots to create the faces, we proceed to calculate these points, all of them contained on the surface of the theoretical sphere requested by the user.

These points, as they are calculated in the order specified above, are stored in that order in the matrix TRI. To carry out this task it will be loaded the coordinates (R, phi, theta) of the points corresponding to the primitive faces (4 in case of the tetrahedron) and n^2 groups of 3 points corresponding to each face will be calculated and stored in the array TRI. To carry out this process it is used an auxiliary matrix called TRI1, which only serves to accumulate and then transfer new information to TRI, without deleting the previous data already stored in it. After the creation of the points of a face it will take place the graphic representation of the triangular face, the coordinates for the next face will be loaded and it will repeat the process with all the primitive faces until the representation of the final polyhedron (tetrahedral, cubic, octahedral or icosahedral approach) is completed. The algorithm for calculating the new sub-faces of each original triangular face needs two nested loops that will sweep each initial triangle from top to bottom and from left to right (fig. 2) by creating new items.

This algorithm uses the coordinates of the vertices of each primitive faces. Since two of these coordinates are angular, in certain faces (they could be called "closure faces " of the polyhedron, since they use the first and last vertices of the same and close it definitely) they coexist vertices belonging near-zero angular coordinates but positive in any case (first points) with other vertices belonging angular coordinates close to 2 * π radians (360 degrees). In these cases the algorithm could give erroneous results, since the calculation of intermediate points provide a range of points between 0 and 2 * π radians, namely, points distributed throughout the whole sphere, without any of them inside the corresponding initial face area, as it should be. To fix this possible programming error, it comes before implementing the algorithm, to approximate the angular coordinates to each other as follows: If a parameter is very high (close to 2 * π radians) and the other is close to zero it is necessary to add 2 * π radians to the lowest one, eliminating the problem without changing the physical value of the variables.

After calculating each new array TRI, as mentioned above, we proceed to the representation of the triangular face that represents, according to the binding order of the points specified in the ORDER matrix. For this task and in order to reduce the structural complexity of the program, it will be programmed a separate function, which is called triangles(ORDER, X, Y, Z). This function takes as parameters the ORDER matrix and the vectors of coordinates of points X, Y and Z. Its only effect is the three-dimensional representation in the MATLAB environment of the various sub-faces generated from each primitive face. The given points in the cartesian coordinates vectors will be joined following the order of union of the vertices specified in the ORDER matrix.

Before invoking this new function, the vector of Cartesian coordinates X, Y and Z are generated from the spherical coordinates of the points contained in the matrix TRI. Only a simple conversion from spherical coordinates to cartesian coordinates algorithm is needed (remember the equations eq. 1, eq. 2, eq. 3; being θ the zenithal coordinate and ϕ the azimuthal coordinate). After the return from the function, writing on the chart is enabled by including the command "hold on", which will overlap more faces in subsequent loop cycles without removing the previous ones or creating a new chart each time. When all the iterations of the loop have been done (as many iterations as faces of the original polyhedron), the command "hold off" is used to disable the writing on the current chart and the program ends.

2.4 Differences between each triangulation algorithm

In the tetrahedral triangulation case it has been decided, in order to distinguish this triangulation from the other methods and take advantage of the fact that the tetrahedron has one side parallel to the ground plane (z = 0) by placing the corresponding ternary axis perpendicular to the ground, not to apply the triangulation to the face contained in this plane. This modification achieves to guide the project, at least one of its branches, to a possible practical application in the design of domes or spherical base covers built on a triangular lattice base. Due to this change, it varies the size of some of the matrices used in the program, as it is evident. The tetrahedral triangulation shows a six-planes-of-symmetry spherical approach as a result. Each plane is defined by each edge and the medium point of the opposite one. This result will have 4 * n^2 faces, being n the accuracy parameter described above.

The fig. 3 shows the result of applying the tetrahedral triangulation algorithm to a sphere of one unit of radius. The value of the accuracy parameter (n) used in this first approach was 6. Another simulation of the same algorithm was carried on using a value of 20 for the “n” parameter. The result is shown in the fig. 4.

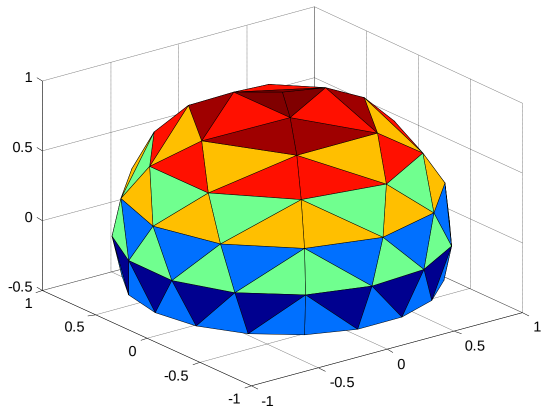

Fig. 3 Tetrahedral triangulation with n=6

Tetrahedral triangulation of the sphere obtained using an accuracy parameter of n = 6 means that each edge of the original tetrahedron (inscribed inside the theoretical sphere which radius is given) is divided in six new edges, generating 36 new triangles per original face.

As it was said above, one of the faces is not triangulated in the tetrahedral approach, so the final number of triangles in the mesh is 108.

This figure and the rest of approaches to the sphere showed below were obtained running the program in the MATLAB console. Although the graphic results of the simulation come from the MATLAB environment, they are only a way to show visually the obtained results (vertices of the generated approach) and they can be easily stored and showed using any other graphic tool.

The colour assigned to each face depends on the value of the Z coordinate (vertical height): Dark blue is assigned to the ones that show a lowest value of the Z coordinate and dark red is assigned to the ones that show a highest value. Points with a value of the Z coordinate between the minimum and the maximum get as colour the result of the lineal combination of the extreme colours.

In tetrahedral triangulation of the sphere obtained using an accuracy parameter of n = 20 (as show figure 4), each edge of the original tetrahedron is divided in 20 new edges, generating 400 new triangles per original face.

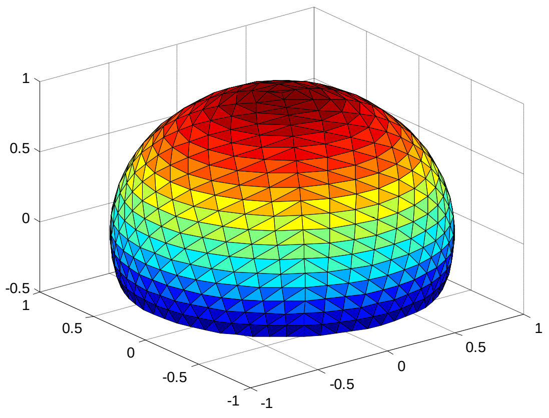

Fig. 4 Tetrahedral triangulation with n=20

Only three faces were triangulated, in order to suggest a real application in the vault calculus, like in the previous simulation result. The final number of triangles in the generated mesh is 1200.

The cubic triangulation produces a spherical approach from a cube, so the triangles are not equilateral and a lower structural quality approach is obtained. Anyway, it is more accurate than the tetrahedral one. This spherical approach will have 6 * 4 * n^2 faces, being n the accuracy parameter described above.

The fig. 5 shows the final polyhedron obtained from the application of the cube-derived triangulation algorithm. The used value of “n” was also 6 in this first simulation and also 20 in the second one, which result is shown in the fig. 6.

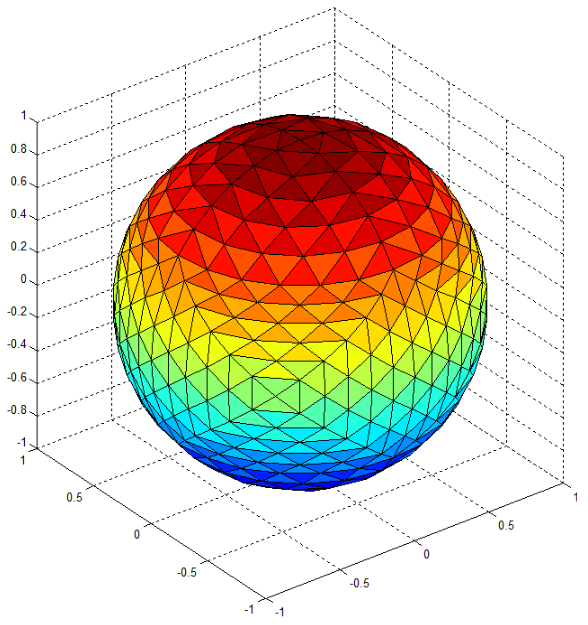

Fig. 5 Cube-derived triangulation with n=6

In the approximation obtained from the cube using an accuracy parameter of n = 6 (Fig. 5), each edge of the polyhedron generated (not edges of the cube) is divided in 6 parts, generating 36 new triangles per triangle. It produces a quite complete meshing, with 864 generated triangles.

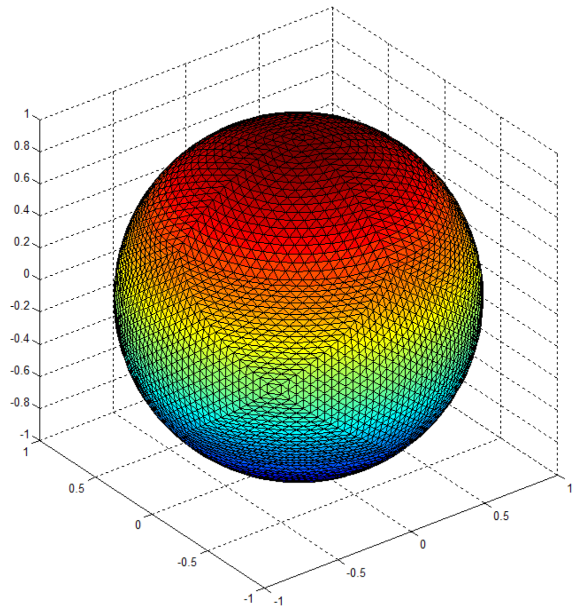

Fig. 6 Cube-derived triangulation with n=20

In the cube-derived triangulation of the sphere obtained using an accuracy parameter of n = 20, each edge of the polyhedron generated from the cube is divided in 20 parts, generating 400 new triangles per original triangle. It produces a 9600 triangles mesh.

The octahedral approach offers a regular structure, made from equilateral triangles, with nine symmetry planes but a worse mesh density than the cubic one, with only 8 * n^2 faces in the final result.

The same two values of the “n” parameter were used to carry on the simulation of the octahedral triangulation algorithm. The obtained result for the n = 6 approach appears in the fig. 7. This simulation produces a quite regular mesh of 288 triangles.

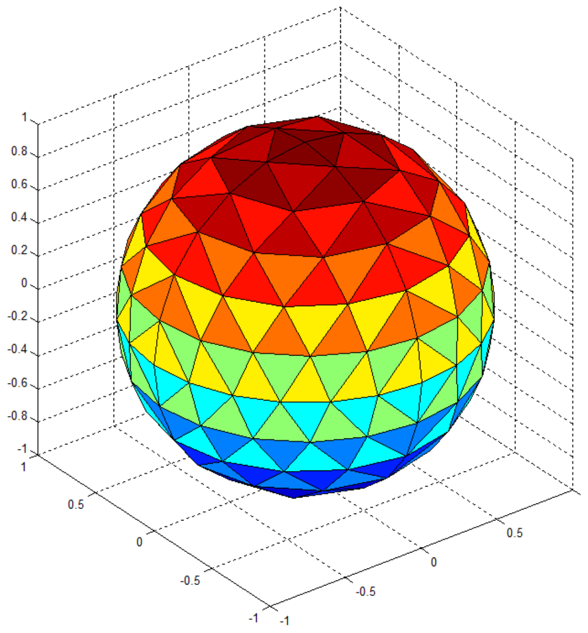

Fig. 7 Octahedral triangulation with n=6

The result for the approach with n = 20 one is shown in the fig. 8. This simulation produces a regular mesh of 3200 triangles.

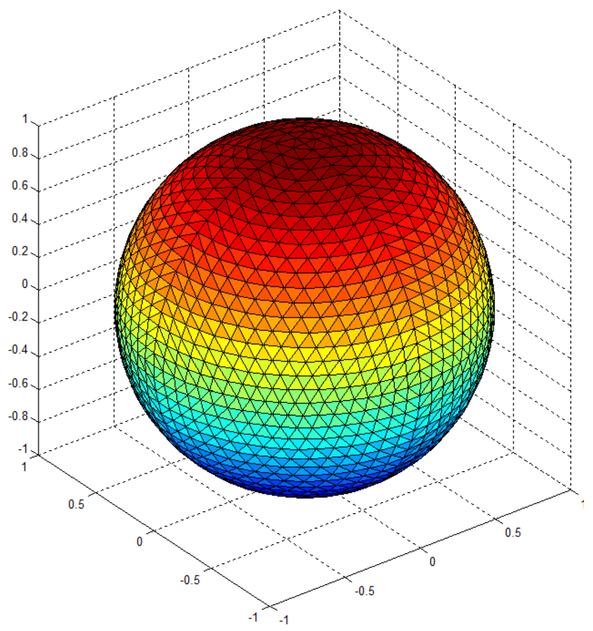

Fig. 8 Octahedral triangulation with n=20

Finally, the icosahedral spherical approach is the alternative that belong the best characteristics. The result has fifteen planes of symmetry and it generates 20 * n^2 equilateral faces following a regular icosahedral structure.

Proceeding like in the previous approaches, the icosahedral triangulation one is tested using the same values for the "n" parameter. The obtained results of the algorithm are graphically represented in the fig. 9 and in the fig. 10.

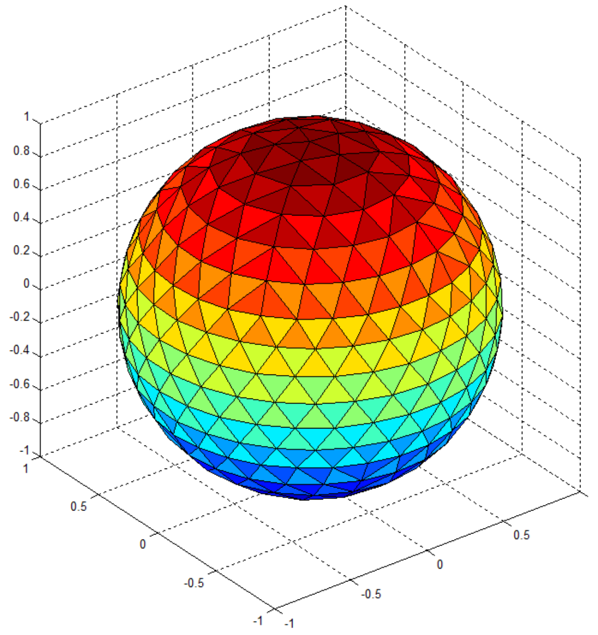

Fig. 9 Icosahedral triangulation with n = 6

The obtained result in this case is a highly regular mesh of 720 triangles.

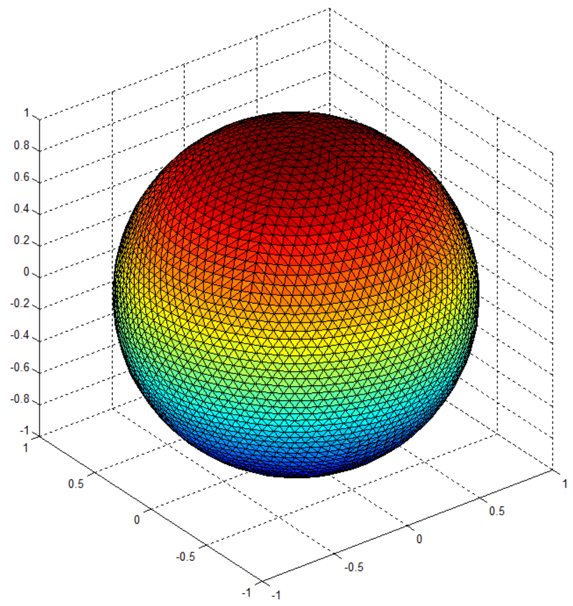

Fig. 10 Icosahedral triangulation with n = 20

The result is a highly regular mesh of 8000 triangles.

A quick comparative overview of the four different methods can be obtained easily from the tab. 2.

Triangulation method	Tetrahedral	Cubic	Octahedral	Icosahedral
Seed body vertices	4	8+6	6	12
Seed body edges	6	12+24	12	30
Seed body faces	4	6*4	8	20
Final polyhedron faces	4*n^2	24*n^2	8*n^2	20*n^2
Accuracy	Low	Medium	Medium	High
Mesh quality	Medium	Low	Medium	High

Tab. 2 Relevant data of each triangulation method

This table shows the number of vertices, edges and faces of each seed polyhedron as well as a qualitative description of the accuracy of each method and the structural quality of the corresponding mesh.

3 Conclusions

3.1 Comparison of the geometric regularity between the MATLAB "SPHERE" approach and the developed approaches

In order to compare the geometric regularity of the different triangulation methods developed and compare it to the results obtained for the built-in MATLAB "SPHERE" approach the divergence between the longest and the shortest edge of each own approach will be calculated.

Since we would like to get values of that divergences for several accuracys (using the number of faces of the approach as an accuracy parameter, for example) of each approach method, a high-level program will be created, in ordre to proccess and show all the divergence data of each approach and it's evolution with the total number of faces generated (N^2).

This program will execute one by one all the sub-programs and obtain the value of the longest and shortest edges generated. The geometric divergence of each method will be obtained as the difference between both parameters. The proccess will be repeated for all the wished values of the accuracy parameter. All the obtained data will be storaged and showed in a graph in order to compare our approaches to the MATLAB one more easily.

The obtained data for several values of the number of faces (N^2, where “N” is not the described accuracy parameter but the square root of the total number of generated faces, to allow the easy comparison with the MATLAB “SPHERE” command obtained results) was collected in tab. 3, where the first row contains the data related to the tetrahedral approach, the second one refers to the cubic-based one, the third one to the octahedral approach and the last one to the icosahedral one.

Tab. 3 shows the results of the spherical approach regularity analysis (divergence between edges) applied over the MATLAB “SPHERE” algorithm. The values are given as multiples of the radius of the original sphere (R).

Faces (n^2)	Tetrahedral	Cubic	Octahedral	Icosahedral
5^2	0.4494	0.2352	0.1801	0.0171
10^2	0.3122	0.2685	0.1924	0.1372
15^2	0.2343	0.2019	0.1559	0.1167
20^2	0.1821	0.1581	0.1279	0.0979
25^2	0.1508	0.1273	0.1089	0.0841
30^2	0.1303	0.1075	0.0945	0.0733
35^2	0.1147	0.0948	0.0837	0.0652
40^2	0.1025	0.0850	0.0748	0.0587
45^2	0.0926	0.0771	0.0679	0.0533
50^2	0.0845	0.0706	0.0621	0.0490

Tab. 3 Results of the spherical approach regularity (divergence between edges)

A quick view of tab. 1 and tab. 3 shows that even the worse triangulation method developed, at least in terms of geometric regularity, which is the tetrahedral one, is much better than the meshing method used by the MATLAB “SPHERE” function. Having a look at the obtained data for 50^2 faces, the MATLAB “SPHERE” approach offers a divergence of almost 25%R, meanwhile the tetrahedral triangulation reaches a divergence of 8.45%R, the cubic-based triangulation 7.06%R, the octahedral triangulation 6.21% and the icosahedral triangulation method is able to reduce the divergence below the 5%R.

The evolution of the divergence increasing the number of divisions done in each of the original edges of the seed polyhedron (real “n” parameter) for each method is easily observable in fig. 11, fig. 12, fig. 13 and fig. 14, which show the collected data of the regularity analysis carried on for the tetrahedral triangulation method, the cubic-based triangulation method, the octahedral triangulation method and the icosahedral triangulation method, respectively.

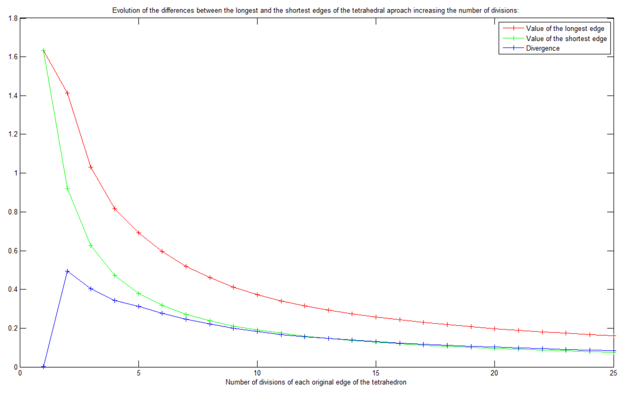

Fig. 11 Divergence between edges (tetrahedral approach)

In Fig. 11 we can see the evolution of the divergence between edges of the tetrahedral approach by increasing the value of the accuracy parameter “n”. The divergence (blue line) is the difference between the longest edge (red line) and the shortest one (green line).

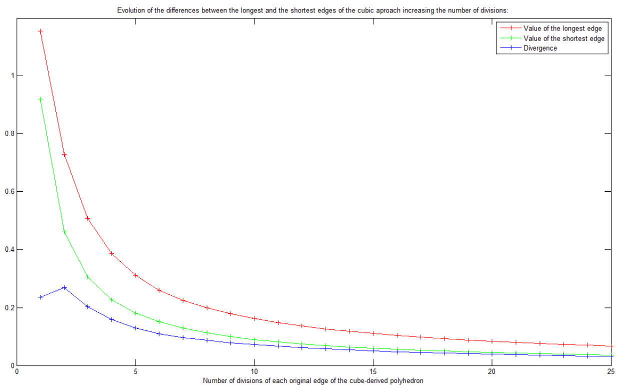

Fig. 12 Divergence between edges (cubic approach)

Fig. 12 shows the evolution of the divergence between edges of the cubic-based approach by increasing the value of the accuracy parameter “n”. The divergence (blue line) is the difference between the longest edge (red line) and the shortest one (green line).

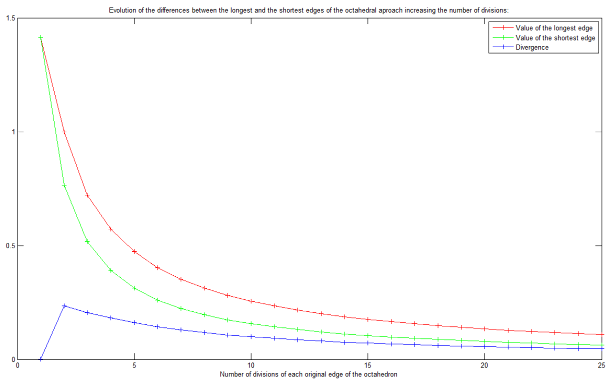

Fig. 13 Divergence between edges (octahedral approach)

Fig. 13 shows the evolution of the divergence between edges of the octahedral approach by increasing the value of the accuracy parameter “n”. The divergence (blue line) is the difference between the longest edge (red line) and the shortest one (green line).

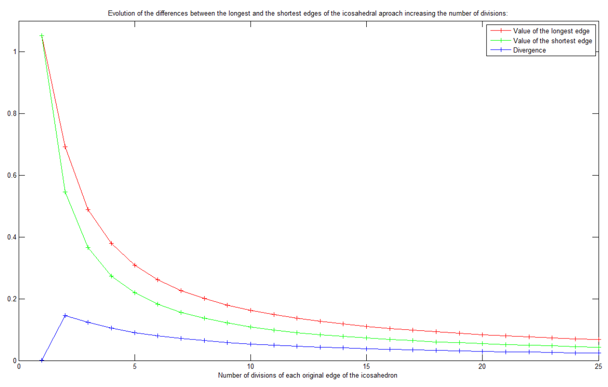

Fig. 14 Divergence between edges (icosahedral approach)

Fig. 14 shows the evolution of the divergence between edges of the icosahedral approach by increasing the value of the accuracy parameter "n". The divergence (blue line) is the difference between the longest edge (red line) and the shortest one (green line).

The supremacy of the icosahedral triangulation method, according to the geometric regularity, over the MATLAB "SPHERE" function and the rest of the developed triangulation methods is observable in these last figures. The starting value of the divergence is extremely low and it falls really fast increasing the value of "n" (divisions per edge of the original polyhedron).

3.2 Comparison between the own algorithms and conclusions about future applications

After to have checked the four algorithms created to give a triangular approach to the sphere it is obvious that the best results are obtained using the icosahedral approach. The accuracy of this triangulation method is extremely high using a relatively low number of triangles. The generated triangles are equal to each other in almost all the cases and the structural regularity of the obtained mesh is also the best one.

This result is easily expected taking into account the fact that the icosahedron is the regular polyhedron that has the highest number of equilateral triangles as faces (20), as the tab. 2 shows. Apart from that, the regular distribution of its faces makes the icosahedron probably the best seed body to carry on an accurate triangulation of the sphere.

The tetrahedral triangulation method reaches also a high accuracy from certain values of the "n" parameter. Probably the main problem of this approach is, more than the low accuracy for low "n" values, the preservation of the three edges of the tetrahedron after the triangulation, as circumference arcs over the theoretical spherical surface.

This "problem" could also be an advantage. Despite the existence of this phenomena mean, first of all, that the triangles close to them are not equals to the rest, it also mean that if this model is used to build architectonical elements it is highly probably that the rest of the bars of the structure discharge their tensions over this other bars. In other words, this effect could create three base points where the whole weight of the structure and the rest of the charges over it will transmit their efforts. Of course this effect will depend on the distribution of charges over the structure. Anyway, it would allow us to distribute the charge over three single points (columns), which is more stable than using four or more supporting points.

The cubic approach generates a higher number of triangles than the expected one for a fixed value of "n" because each face of the original cube is converted, before starting the application of the algorithm, in four non- equilateral triangles. The high number of faces doesn't mean a high quality mesh because the cubic base and the later adaptation to a triangular-faced polyhedron make some convergence points to appear. These points are placed in a position corresponding to the centre of each cubic face, common point between the transformed edges.

This characteristic can be also used in architectural elements design and building because it allows us to build semi-spherical domes with four points of tensional concentration, where the columns must be placed.

The octahedral triangulation method offers a better approach from the mathematic point of view, eliminating divergences between triangles and convergence points. The obtained accuracy is higher for the same number of faces in the final polyhedron. It's the second best model, after the icosahedral one, of course.

The icosahedral triangulation algorithm offers the best result and it could have thousands of applications, not just in the design and development of domes, but in any area of the applied sciences where a high accuracy discrete model of the sphere is needed. The softness and regularity of the obtained result is clear in the fig. 9. The accuracy of the approach can be increased as much as needed due to the high computational efficiency of the generated algorithm.

The only problem could be the computing time but to reach higher than one second computing times it was necessary to use values of the accuracy parameter (n) around 100. It means 200000 triangles in the icosahedral approach final polyhedron.

It was impossible, for now, to make any of these algorithms to fail or break the execution of the program.

Acknowledgement

To colleagues of Graphic Expression in Engineering, always willing to cooperate selflessly.

References

[1] M. Berger. *The space of spheres*. In Geometry I, Springer 1987, pp 349-361.
[2] P.M.M. De Castro, F. Cazals, S. Loriot, M. Teillaud. *3D spherical geometry kernel*. CGAL User and Reference Manual. CGAL Editorial Board, 3.5 edn. 2009.
[3] M. Caroli, M. Teillaud. *Computing 3D periodic triangulations*. ESA, 2009.
[4] S. Pion, M. Teillaud. *3D triangulations*. CGAL User and Reference Manual. CGAL Editorial Board (ed.), 3.5 edn. 2009.
[5] Library for Efficient Data Types and Algorithms (Leda) website. http://www.algorithmic-solutions.com/enleda.htm Accessed 19 Mar 2010.
[6] J.D. Boissonnat, M. Yvinec. *Algorithmic Geometry.* Cambridge University Press, UK, 1998.
[7] K.Q. Brown. *Geometric transforms for fast geometric algorithms.* Report CMU-CS-80-101, Dept. Comput. Sci., Ph.D. thesis, Carnegie-Mellon Univ. Pittsburgh, PA. 1980.
[8] S.L. Chan, E.O. Purisima. *A new tetrahedral tesselation scheme for isosurface generation.* Computers

& Graphics, Volume 22, Issue 1, 25 February 1998, pp 83-90.
[9] D. Goldberg. *What every computer scientist should know about floating-point arithmetic*. ACM Computing Surveys, 23 (1), pp 5-48. 1991.
[10] C. Li, S. Pion, C.K. Yap. *Recent progress in exact geometric computation.* Journal of Logic and Algebraic Programming, 64 (1), pp 85-111. 2005.
[11] C.K. Yap, T. Dubé, *The exact computation paradigm.* Computing in Euclidean Geometry, 4, pp 452-492. 1995.
[12] F. Haußer, Y. Luchko*. Mathematische Modellierung mit MATLAB: Eine praxisorientierte Einführung.* Spektrum Akademischer Verlag Heidelberg, 2011.
[13] A. Gilat, V. Subramaniam. *Numerical Methods for Engineers and Scientists: An Introduction with Applications Using MATLAB, 2e.* John Wiley & Sons, Inc. 2011.
[14] W. Peng. *Efficient Programming Techniques and Applications in MATLAB: 25 Case Studies.* BUAA Press, 2010.
[15] C. F. Van Loan, K.-Y. Daisy Fan. *Insight Through Computing: A MATLAB Introduction to Computational Science and Engineering.* SIAM, 2010.
[16] V. Rovenski*. Modeling of Curves and Surfaces with MATLAB.* Springer, 2010.
[17] I. Danaila**,** P. Joly**,** M. Postel**,** S. Mahmoud Kaber*. An Introduction to Scientific Computing: Twelve Computational Projects Solved with MATLAB.* Springer, 2007.
[18] A. Quarteroni, F. Saleri. *Cálculo Científico con MATLAB y Octave.* Springer, 2007.
[19] G. Strang. *Computational Science and Engineering.* Wellesley-Cambridge Press, 2007.
[20] G. Gan, C. Ma, J. Wu. *Data Clustering: Theory, Algorithms, and Applications.* SIAM, 2007.
[21] M. C. Ferris**,** O. L. Mangasarian**,** S. J. Wright. *Linear Programming with MATLAB.* SIAM 2007.