Quadtrees on the Sphere

C. Godsalve
email: seagods@hotmail.com

October 1, 2008

Contents

1 Personal Note

A long time ago, I had used a recursive subdivision on a truncated icosahedron to generate a triangular mesh on a sphere. The truncated icosahedron is the structure of C60, also named Buckminsterfullerene after the architect BuckMinster. It consists of pentagons and hexagons, so is an Archimedean solid, and is familiar across the word as the modern soccer ball. This was a tiny sub=problem on the project I was immersed in at the time.

I recently returned to this problem, and decided that the approach was too complicated, then I thought I had a neat idea about quadtrees on a sphere based on an octahedron, and implemented it in C++.

Recently, on going through a pile of papers I hadn't looked at for for a decade, it turns out to be exactly the idea used by Goodchild and Yang [1]. Indeed the basic idea here goes back further to Dutton [2] and probably back further still. It seems as if my neat idea was an unconscious memory of having a quick glance at a paper, shoving it in a "read later" draw, and forgetting about it.

I didn't quite like the resulting large variance in size and shape that this approach gives rise to, and thought that a similar approach, but based on the icosahedron might improve matters, However it turns out that Fekete published this idea in 1990 [3].

In both methods, each triangular face is split into four sub-triangles, each of which is split into another four sub-triangles, and so on. As a basic idea, it is far from complicated, but there are many details to consider on how the triangulation is ordered, how to encode triangle neighbours, and some subtle points that arise regarding uniformity of coverage. We address these in this article.

If, with the icosahedron, we had split the first level into nine subtriangles with two extra nodes on the edges of the original, we get exactly what we would have with the truncated icosahedron. So, qualitatively, the icosahedron subdivision here gives the same result as my earlier method, and is far simpler.

I have written this article for my own notes, and it isn't quite yet completed. My own code OctaSphere.cpp and IcoSphere.cpp are just straightforward implementations using pointers, I have not even got to coding in any functionality, and I have not included such useful things as binary encoding of quadtree cells. So, here it is...

2 Introduction

This article is about a tessellating a sphere with a triangulation, or with many triangulations on different scales in the form of quadtrees. Using a local triangular mesh can sidestep many complications that may result from global spherical coordinates.

For instance, we may want to use a triangular mesh for solving partial differential equations on a sphere. or indeed on some much more arbitrary surface. The sphere may be deformed into some other shape. As long as the any vector from the origin points to one (and one only) point on the surface, the triangulation on the sphere will provide a triangulation on the deformed shape. Whether of suitable quality for whatever purpose is shape and purpose dependent.

So, it will always work for a convex surface. If the surface is not convex there will be points where our look direction from a given origin "skewers" the surface more than once. However we might be able to change the origin so that surface is single valued seen from that position. This is by no means always the case, but nonetheless a large variety of surfaces may be amenable to the present approach.

3 Octahedron, Icosahedron, or Other?

If we are considering the sphere, the octahedron automatically gives us two main meridians, the poles, and the equator. From this point of view it is very attractive. At first sight, the icosahedron seems to introduce many complications. We shall see that the the problems introduced with the icosahedron are easily surmountable.

Our ideal tessellation of the sphere would be one where all the triangles had the same shape and size. Equilateral triangles would be wonderful. This is impossible, the icosahedron is the platonic solid with the largest possible number of facets, and that's that.

Away from the corners of the initial shape, the subtriangulation consists of irregular hexagonal cells made up of six triangles. That is to say, every node occurs in six triangles, so that the triangles attached to the node from a hexagon. At the corners, the original number of facets attached to the corner node stays the same. So three will be three triangles attached to the corner nodes for the tetrahedron and the cube. So, we shall rule out a cube split into sub-triangles and the tetrahedron as they will introduce a great number of triangles that are far away from being equilateral. This leaves only the octahedron and icosahedron, having four and five triangles attached to the corner nodes respectively.

So, the octahedral option as in Dutton [2] and then Goodchild and Yang [1], and the icosahedral option [3] are the only options to be studied here.

4 Options for Generating the Mesh

Before continuing, we shall look at some fairly obvious trigonometry. However simple it may be it shows us there are three different approaches to getting from the base shape to the sphere.

Every line segment that we shall split in half shall be the chord of some great circle. In this section, we shall speak loosely of longitude, though what we shall really mean is the angle measured from some arbitrary point point on the great circle on which our line segment lies.

In Fig.1, we see a chord NQ of a segment of a circle of half angle θ. We suppose NQ to

be a level zero line segment. The local coordinate α_0 varies from zero at N to 1 at Q.

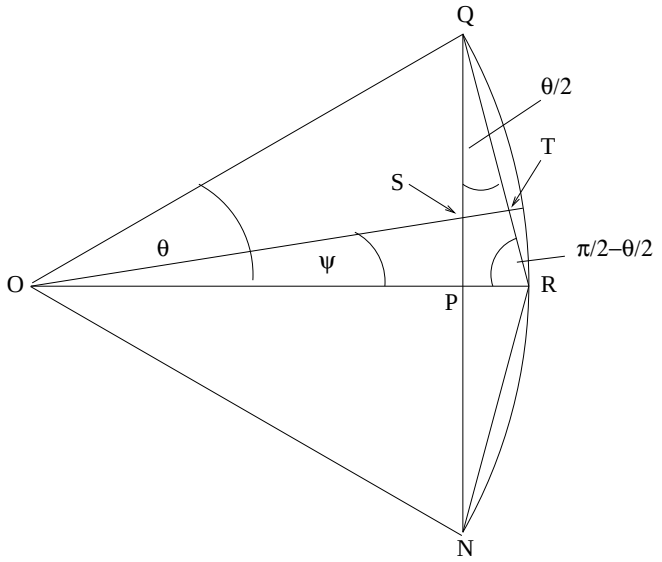

Figure 1: At Level 0, we have NQ, at level 1 we have PQ and/or RQ. What are the local coordinates of the same longitude ψ_1 on segments PQ and NQ?

We now suppose that we split this line segment in half at P to form two level 1 segments. This can be done in two ways. We may split NQ and place a node at P, or we can split it, and then project the point P to R.

Suppose we find the projection point of the angle ψ onto NQ, and suppose also that as we subdivide NQ, all the subsequent recursive subdivision points remain on NQ. We can easily follow what the local coordinate in the new subdivision is from the previous subdivision's local coordinate of that point. So, if the coordinate at level 0 is α_0, then the coordinate at level 1 will be $2 \times (\alpha_0 - 1/2)$ if $\alpha > 1/2$ and $2 \times \alpha_0$ otherwise. We see how to do this for the triangle in §6.

We can then project all the points on NQ and the endpoints of the sub-intervals onto the great circle at the end of the process. Note, that if we evenly divide NQ, then the divisions on the great circle will not be at equal angles. We call this method a "type-1" projection.

If we subdivide NQ into two, and then project P to R, and do this with every further subdivision, we end up with a set of line segments of equal length with the endpoints on the great circle. We end up with part of a regular polygon. This has the advantage of even coverage. However, there is now a complication, in that new α is not as simply related to the previous α as above.

For example, a line at an angle $\psi_1 \leq \theta$, cuts PQ at S, and RQ at T. Then

$$\alpha_1' = PS/PQ = \tan\psi_1/\tan\theta, \tag{1}$$

is the local coordinate for that longitude for ψ_1 on PQ. If $\theta = 45°$ and $\psi_1 = 22.5°$, then $\alpha_1' = 0.4106$ rather than 0.5. Also

$$\alpha_1 = RT/RQ = \frac{\sin\psi_1}{2\sin(\theta/2)\cos(\psi_1 - \theta/2)}. \tag{2}$$

is the local coordinate for the same longitude on RQ. Note that eqn.2 is exact at the end points *and* for $\psi_1 = \theta/2$. So, we can tell which half to put a given angle in directly from the angles of the end points. We call this new type of projection "type-2".

Because eqn.2 is exact at the end and mid-points, the position vectors of the new nodes are trivial to calculate. (This would not be the case if we were splitting the edges into thirds for example.)

There is yet another method we can use. The value of α is a measure of the geometrical distance of the intersection of the direction vector with the line segment. However, there will be some nonlinear mapping $\alpha \rightarrow \gamma(\alpha)$ such that the γ is a direct measure of the fraction of the arc between the two endpoints. The thing is, we shall project this mapping onto the arc anyway, so we do not even have to do the transformation. All we need to be satisfied is to know that such a mapping exists, and that the inverse exists. So, all we need to do is to *define* α to be linear in ψ. We shall call this method "type 3".

This is the method used in [1]. It has the advantage that we end up with lines of constant latitude. In this method, we immediately know the local coordinates of a point in the new subdivision from the old local coordinates in the same way as type 1. Goodchild and Yang use the odd phrase "we assume x to be linear in longitude". Of course x is not linear in longitude, they do not assume it to be so, and they are not making an approximation that it is so. They are in fact using the method we have called "type 3".

To sum up, the first two types we have map the sphere onto a polyhedron. In the type-1 mapping, the sphere is mapped onto either an octahedron or an icosahedron, and once the local facet coordinates are known, it is trivial to work out the local triangular coordinates from the coordinates at the previous level of subdivision.

In the type-2 mapping, the new position vectors at the midpoints are normalised at the end of the new level of refinement. subsequently, we map the sphere onto a higher order polyhedron at every level. This has the disadvantage of adding complexity, but improves the uniformity of shape of the triangular mesh on the sphere.

Lastly, we have the mapping which is simplest to use, which we call type-3 in this article. It sidesteps the actual polyhedron in an elegant manner, and moves directly to the sphere. However, the underlying structure of the basic polyhedron still shows up in the mapping.

This is the simplest mapping to use, but as we shall see, it has some disadvantages in terms of uniformity in shape.

5 The basis Octahedron and Icosahedron

The basis for our initial data structure and geometry shall be the octahedron as shown in Fig.2. We shall consider the icosahedron later. Each facet shall be have nodes 1, 2, and 3 counterclockwise when viewed from the outsides. Each facet has a coordinate system with the origin at the first node listed in the Fig.2. Edge 1 shall be from 1 to 2, edge 2 shall be from 1 to 3, and edge 3 shall be from 2 to 3.

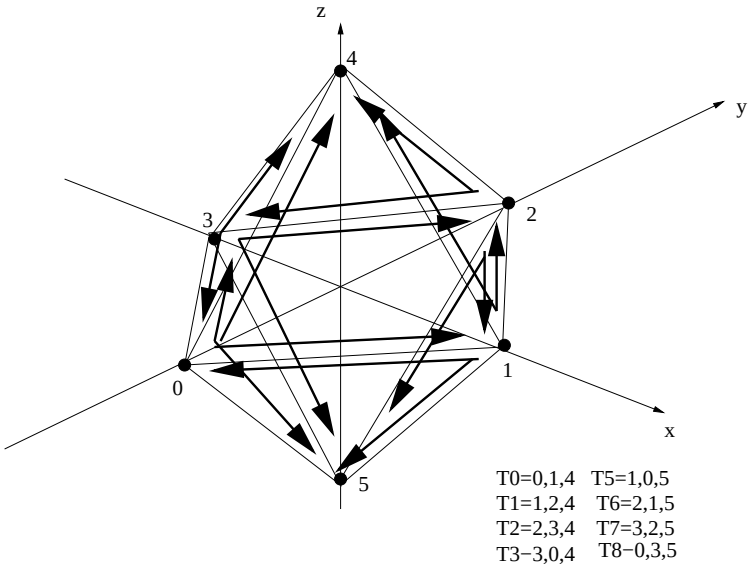

Figure 2: The basis octahedron with nodes $0 = (0, -1, 0)$, $1 = (1, 0, 0)$, $2 = (0, 1, 0)$, $3 = (-1, 0, 0)$, $4 = (0, 0, 1)$, $5 = (0, 0, -1)$

So, any position vector can immediately be assigned to an octant (Triangles 0 to 7) and a quadrant. We shall look at how we can subdivide any individual octant. We concentrate on $T0$ to start off with.

$T0$ is subdivided into four triangles by introducing nodes half way along the edges. The internal triangle is labelled zero, and the four outer triangles are labelled 1 to 3. The internal triangle's coordinate system is "rotated" through 180° with respect to the other three triangles. Each triangle of the subdivision can then be subdivided in exactly the same

way as depicted in Fig.3.

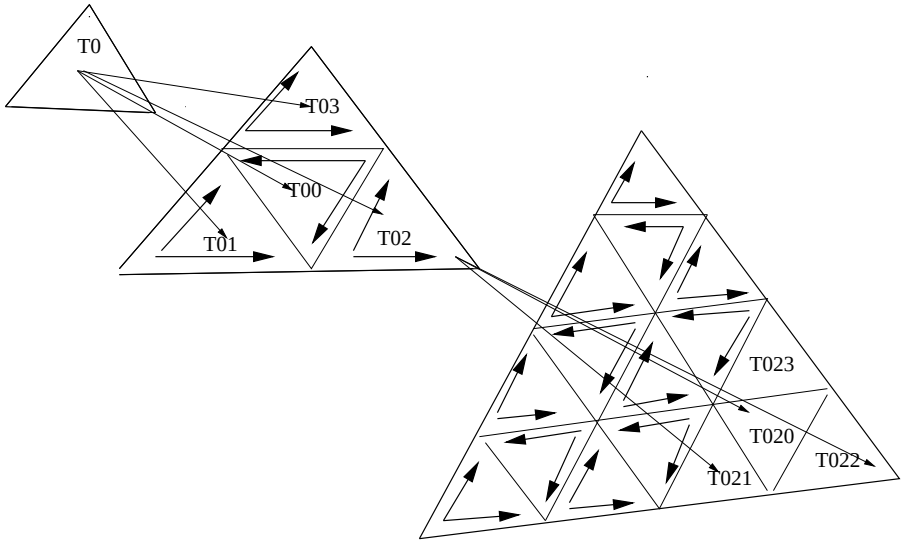

Figure 3: $T0$ points to four sub-triangles, each of which points to four more, and so on.

As a basis for the computer program Sphere.cpp we introduce a Triangle class. Each Triangle has three integers for node numbers. The first is the origin node, and they run counter clockwise when viewed from outside the octahedron. From these numbers we have an internal coordinate system. Edge 1 is node 1 to 2, Edge 2 is from node 1 to 3. On top of this there are two pointers S and N. These are NULL if the triangle has not been split. If the triangle has been split, S points to four integers. The first is the internal triangle of the subdivision. The next three run counter clockwise from the origin of the original triangle. This gives us a basic Quadtree structure as seen in Fig.3. On top of this, we store some extra information. The Triangle shall "know" which triangles lie across edge 1, edge 2, and edge 3.

So, we start off with eight triangles. We visit each triangle in turn and split it and the

subdivision at a partial stage is seen in Fig.4. Here we start off at level zero. We see that

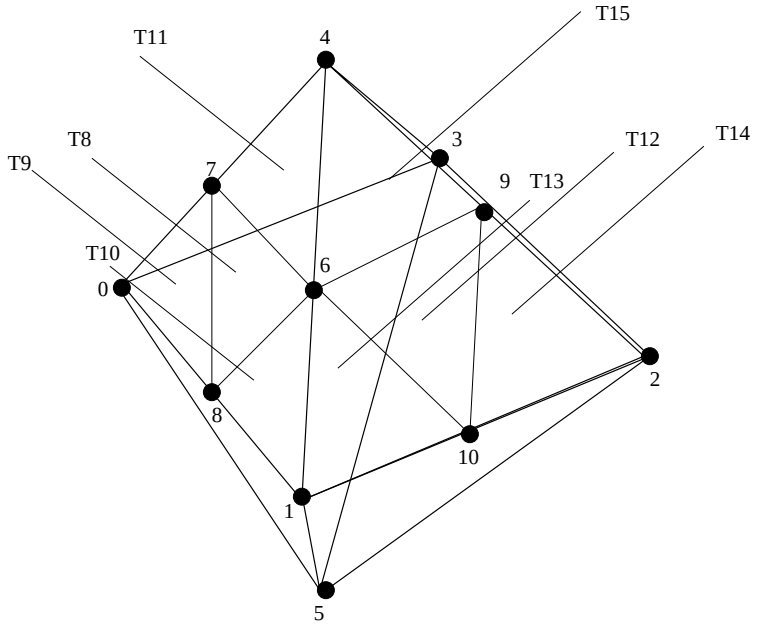

Figure 4: The start of subdividing the root level.

$T0$ has NULL for the value of S. It has not been split. We check that $T0$ has neighbours, and that each of these has NULL values for S. None of the neighbours was been split. We add three new vectors to the list. These are at the positions half way along the original edges. They are numbered 7, 8, and 9. Node 7 is the origin of the root triangle at the next level. We now have triangles 9 to 12. Edge 1 of $T8$ has neighbour $T11$ and edge 1 of $T11$ has neighbour $T8$. Edge 2 of $T8$ has neighbour $T10$ and vice versa. Edge 3 of $T8$ has neighbour $T9$ and vice versa.

We now visit $T1$. We see that $T1$'s S pointer is null, but its neighbour on edge 2 has been split. This means that node 2 of the new root triangle already exists. We introduce node 9 as the origin of the new root triangle, and node 10 as the root triangles node 3. We immediately come across the fact that, if a neighbour of the current triangle ($T1$ in this case) is not in the same quadrant, things behave differently as far as connections are concerned.

Normally, things behave as in the Fig.5. We have an unsplit triangle, which we want to split into 4 red triangles. Any pre-existing triangles on the same level as the "triangles to be" are coloured blue. The triangles neighbouring the current triangle to be split all have split neighbours. Their sub-triangles are blue.

Red triangles 2 and 4 have edge 2 connected to edge 2 of blue triangles 4 and 2. The edges in the neighbours have opposite senses. Red triangles 3 and 4 are connected to blue

triangles 4 and 3 on edge 3 (again in the opposite sense. On edge 1, red triangles 2 and 3 are connected to edge 1 of blue triangles 3 and 2. Again, the edges have the opposite sense in the neighbour.

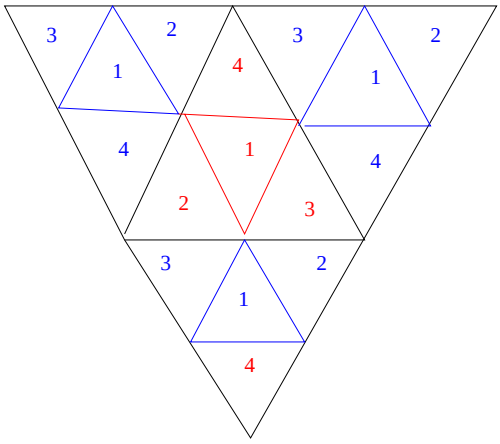

Figure 5: Neighbours in the same quadrant.

Suppose we arrive at a parent triangle to be split that shares edge 2 with a neighbour another quadrant. We see how this is different in Fig.6. This time, split triangles (or child triangles) 2 and 4 have edge 2 crossing into triangles 3 and 4 of the current triangle's left neighbour's "children". These are edge 3 of the neighbour's children, and they have the same sense as the edge 2's of the triangles children to be. A similar situation arises if the current triangle has a split neighbour in the quadrant to the right as depicted in Fig.7. Now the children to be will share edge 3 with edge 2 of children 2 and 4 of the current triangles neighbour 3. However, the way we have ordered things means that the neighbours of child subtriangles 2 and 3 always share edge 1 (in the opposite sense) with child triangles 3 and

2.

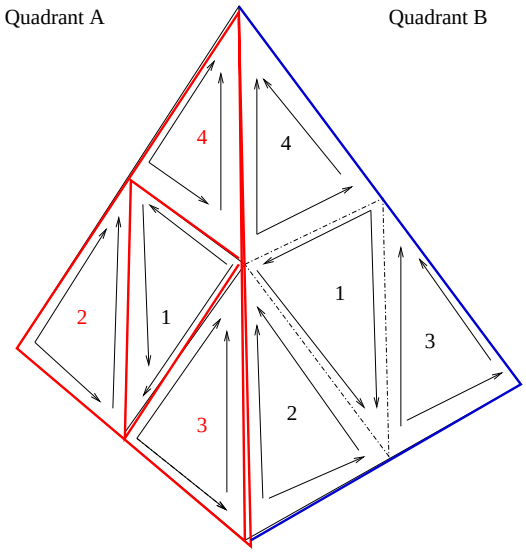

Figure 6: Split Neighbours in the quadrant to the left.

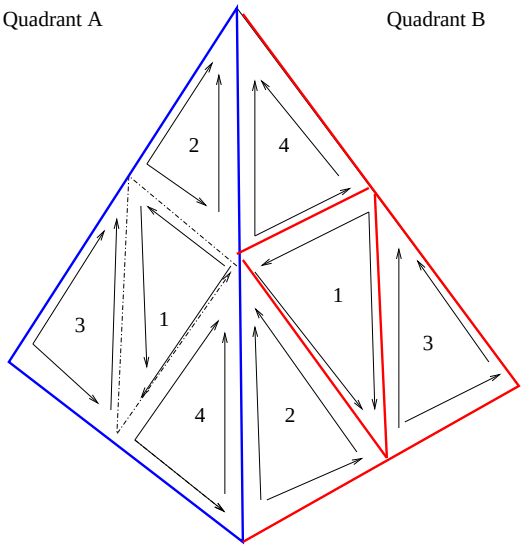

Figure 7: Split Neighbours in the quadrant to the right.

We now move on to the icosahedron. We order the triangles of a basis icosahedron as in Fig.8. The way we have ordered the coordinate systems in each facet mean that the

neighbours and edges in the "polar cap" triangles can be dealt using the same method as was used for the octahedron. The polar cap triangles have edge 1 connected to edge 1 of an "equatorial triangle" (in the opposite sense), and edges 2 and 3 are connected to edge 2 and 3 if the neighbours (again in the opposite sense). In this way, we can proceed as we did with the octahedron.

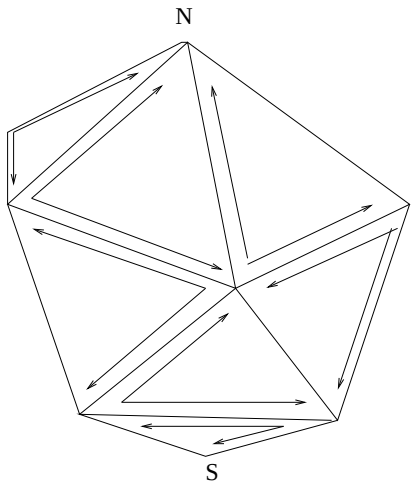

Figure 8: The basis icosahedron is chosen to have nodes at the poles, resulting in two pentagonal rings at constant latitude. Lines of constant longitude radiating from one pole bisect the triangles they cross until reaching the other pole.

6 Where in the World Are We?

On the octahedron, given a latitude and longitude of some point, We know the quadrant straight away. In §2, we saw that if we had $\alpha'_0 = \alpha_0 = 0.705$, then this would correspond to $\psi = 45°$, and $\alpha'_1 = 0.401$, but α_1=1/2. So, if we use a type-2 method, we cannot even use the fact that $2 \times (\alpha'_0 - 1/2) < 1/2$ to decide which half of the line segment RQ the longitude lies on. The following applies to type-1 and type-3 methods.

As we refine the original triangle we wish to keep track of the local coordinates of this point at every new level in the quadtree. Now we have α and β. Consider Fig.9. We immediately "know" from the inequalities for α and β whether this point is in triangles 2, 3, or 4. If it isn't in any of these it is in triangle 1. Moreover, we can immediately calculate the new coordinates (α', β') of our position vector in the sub-triangle. If it is sub triangle 2, then $\alpha' = \alpha/2$, $\beta' = \beta/2$. If it is in subtriangle 3, then $\alpha' - \alpha - 1/2$. If it is in subtriangle

4, then $\alpha' = \alpha$, and $\beta' - \beta - 1/2$ Lastly, we might be in triangle 1, if so, then $\alpha' = 1 - 2\alpha$, and $\beta' - 1$.

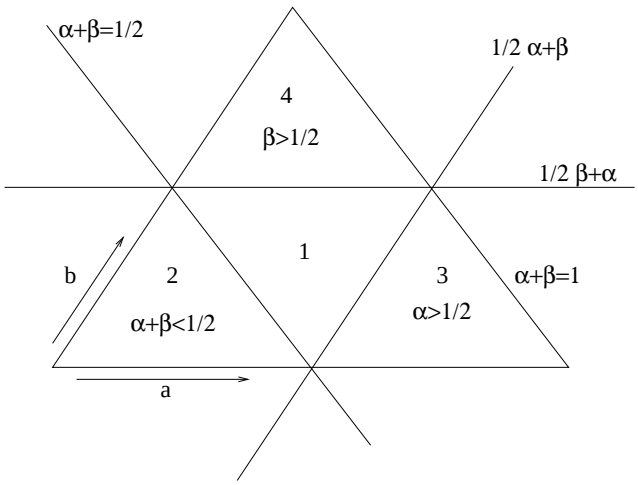

Figure 9: Inequalities to find the child triangle we are in.

Given the local triangle coordinates it is trivial to calculate the latitude and longitude of any point in the subtriangle at the next level.

7 Traversing the Mesh

Consider this problem. As we have jumbled up all the node and triangle numbers, how can we do such things as visit each node or triangle at a given level in turn. We shall assume that the full mesh exists, this might not happen for data storage where there are featureless areas. In this case, to save recall time and storage space, these featureless areas will have the data stored at a low subdivision level, whereas fine detail will be stored at the largest subdivision. This makes things more complicated. So, if at the last level of subdivision, we have all possible triangles set, we just do the following.

Go to level zero, get child 2, get child 2 of child 2, and so on till we get to the highest level we we want to do this for. We are now at the lower left hand corner of the original level zero triangle. We shall call this $Tstart$. We Get neighbour 3 of $Tstart$, that is $N3$ and then set $TstartNext$ to be $N1N3$, by which we mean neighbour 1 Neighbour 3. Now for the nodes. The first node is node 1 of $Tstart$, the second is node 2 of $Tstart$. Next we cross into $N = N2N3$ and the next node is Node 2 of N, Then we cross into $N- > N3N2$

of the current N and add node 2 of that triangle to the list, and so on til we reach the end. Then we set $Tstart = TstartNext$, and set the new $TstartNext$ in the same way we set the first $TstartNext$. We leave out the small details on what to do at the edges and corner. For some applications, we may need to have a list of all the triangles attached to a node, and a list of all the nodes in the outer ring. Mostly these shall form a hexagon.

The possibilities are seen in Fig.10

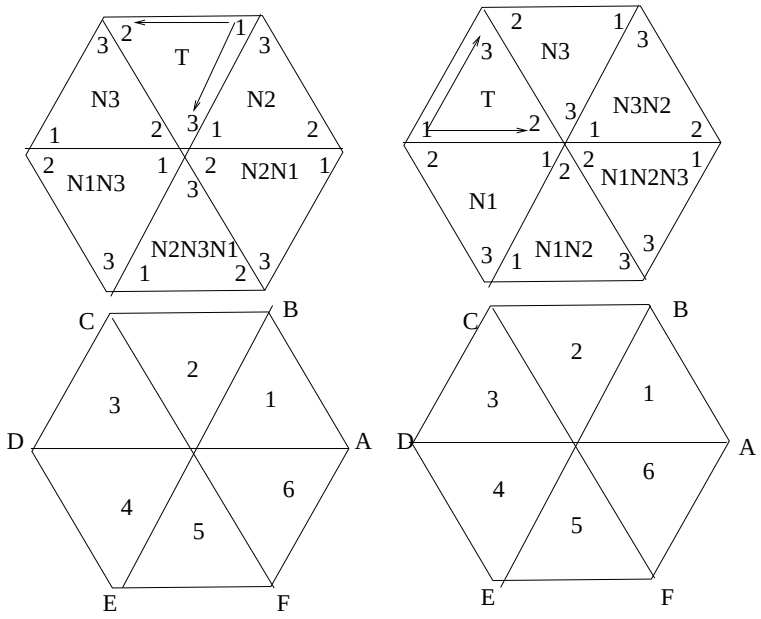

Figure 10: The hexagon surrounding T node 2 and T node 3.

8 Results and Discussion

The level of subdivision is L, with $L = 0$ for the octahedron and icosahedron. The total number of triangles (including all levels) is $N \times (4^{L+1} - 1)/3$ triangles, where N is either 8 or 20. So for the octahedron at L=1, we have $8 \times (16 - 1)/3 = 8 + 32$ triangles in total. The total number of nodes in for the octahedron is $4 \times (4^L - 1) + 6$. For the icosahedron, we end up with $10 \times 2^L + 2$ nodes.

We start off by looking at the type-3 decomposition of the sphere using the octahedron as a basis. In this type of decomposition, we must remember that the poles are singularities where longitude is undefined. So if node 3 of any triangle is equal to 4 or 5, we assign the longitude of the split edges 2 and 4 to be the longitude at node 1 and 2. For the icosahedron, we do the same if node 3 of a triangle is 0 or 11. Also, we must take care of the quadrants for longitude differences.

In Fig.11, we see the triangulation at level 2. The decomposition of the $T0$ level 1 triangle crowds out the four child triangles of $T1$ at level 1. Generally, we have large variations in size and shape.

In Fig.12, we see the level 4 decomposition. The lines of equal latitude are now clear, but the coverage is uneven. The $T0$ triangle's children are large compared to the others, and there are always right angle triangles at the corners of the octahedron. In Fig.13 and Fig.14, we see that there are similar problems at level 2 and 4 for the icosahedron, though the range is size and shape isn't quite so large.

In Fig.15, we see the level 2 subdivision of an octahedron of a type-1 (blue) and type-2 (black) mesh. The type-1 method gives a larger variation in size, but the shape variation is fair. For the icosahedron, we expect the differences to be smaller. We no longer have lines of equal latitude, as all the edges map onto great circles.

As far as locating which triangle a given latitude and longitude is in, using the same allocation as a type-3 mesh will work most of the time. When it doesn't, the local coordinates α and β will indicate a point outside of the triangle, and the particular values which neighbour the point is in. A similar procedure may be used for subdivisions.

Fig.16 and Fig.17 show level four meshes for the sphere using the type-2 method. We conclude that, if uniformity of size and shape are important, then the type-2 decomposition of the icosahedron is the best option, and that the type 1 decomposition will be nearly as good and simpler. If the relative sizes and shapes of the cells don't matter too much, then the simplicity of the type-3 method on the octahedron is the best option, and this option has the added attraction of lines of equal latitude.

The basic programme subdivision programs can be obtained from the author.

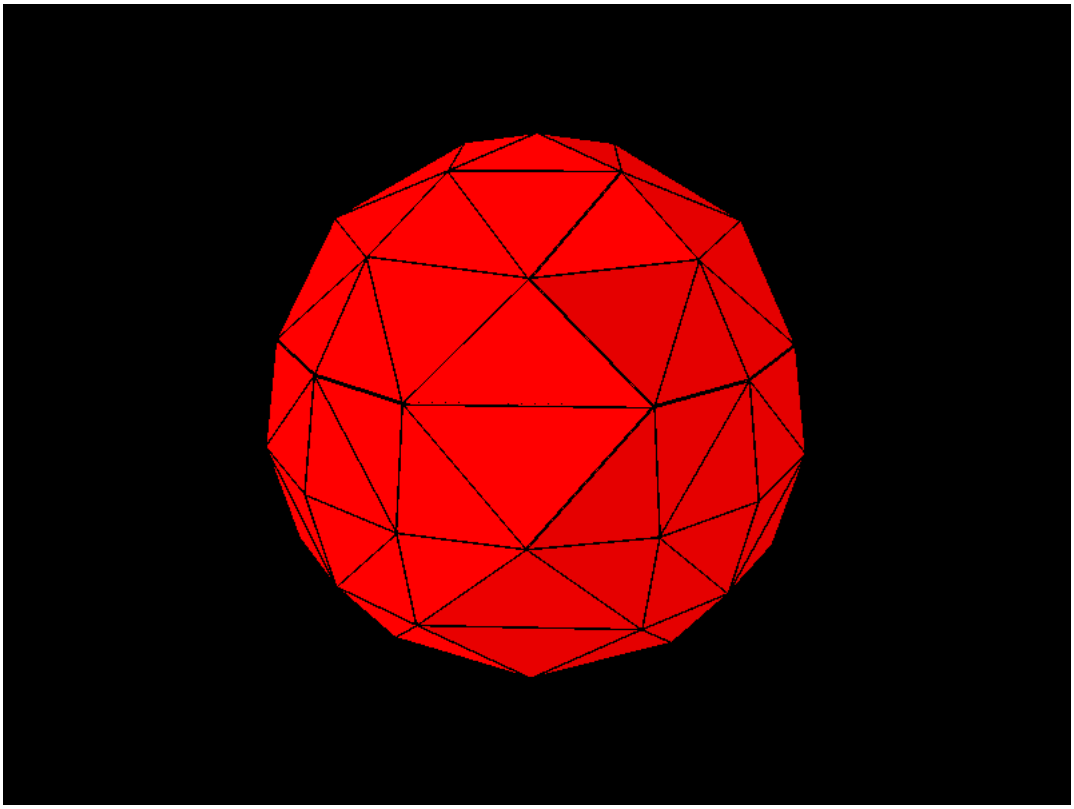

Figure 11: Type-3 subdivision of an octahedron at level 2

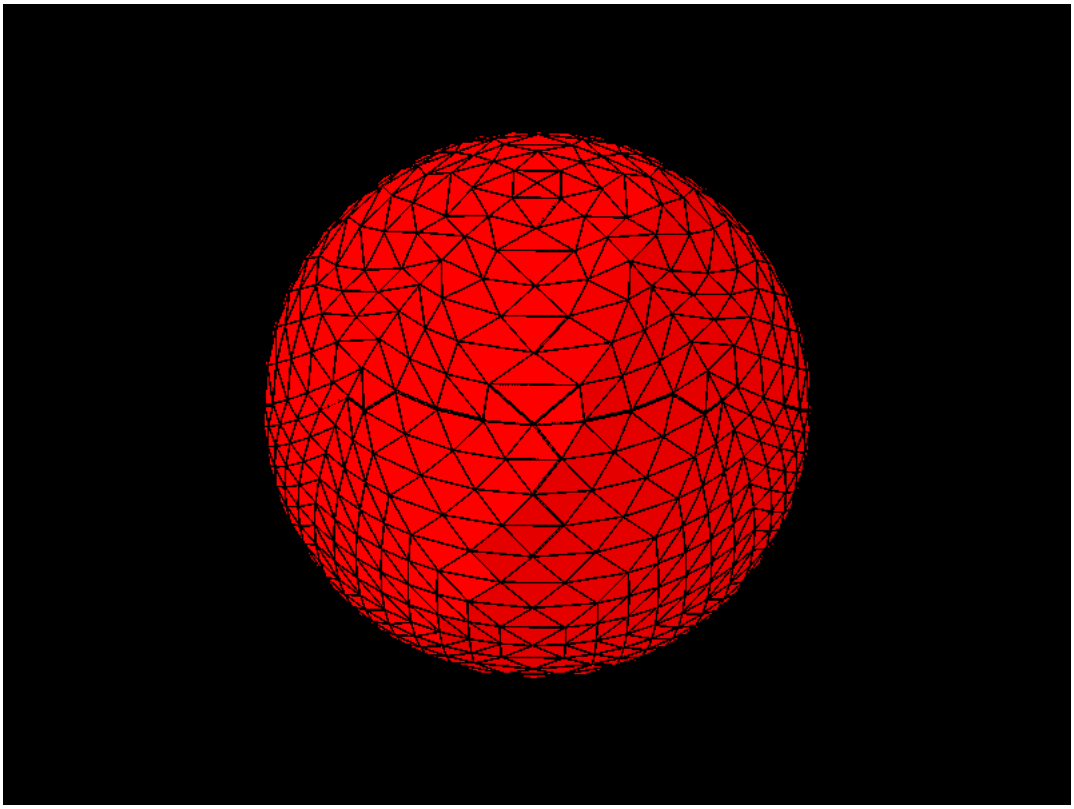

Figure 12: Type-3 subdivision of an octahedron at level 4

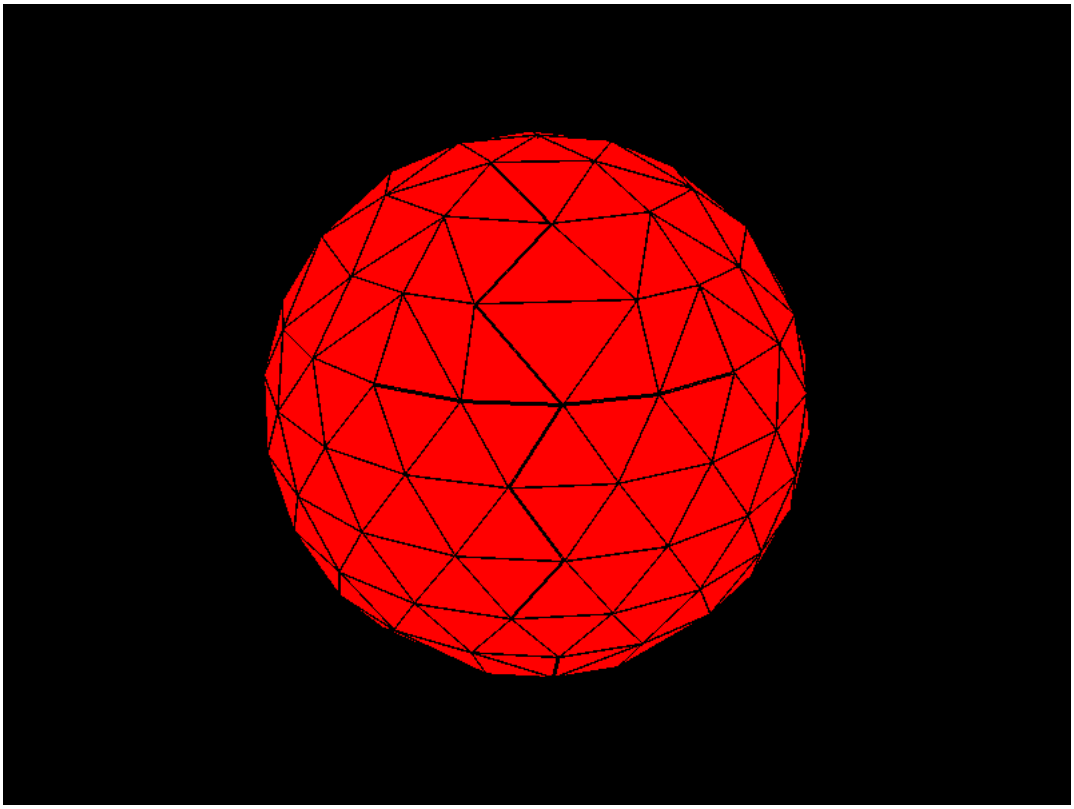

Figure 13: Type-3 subdivision of an icosahedron at level 2

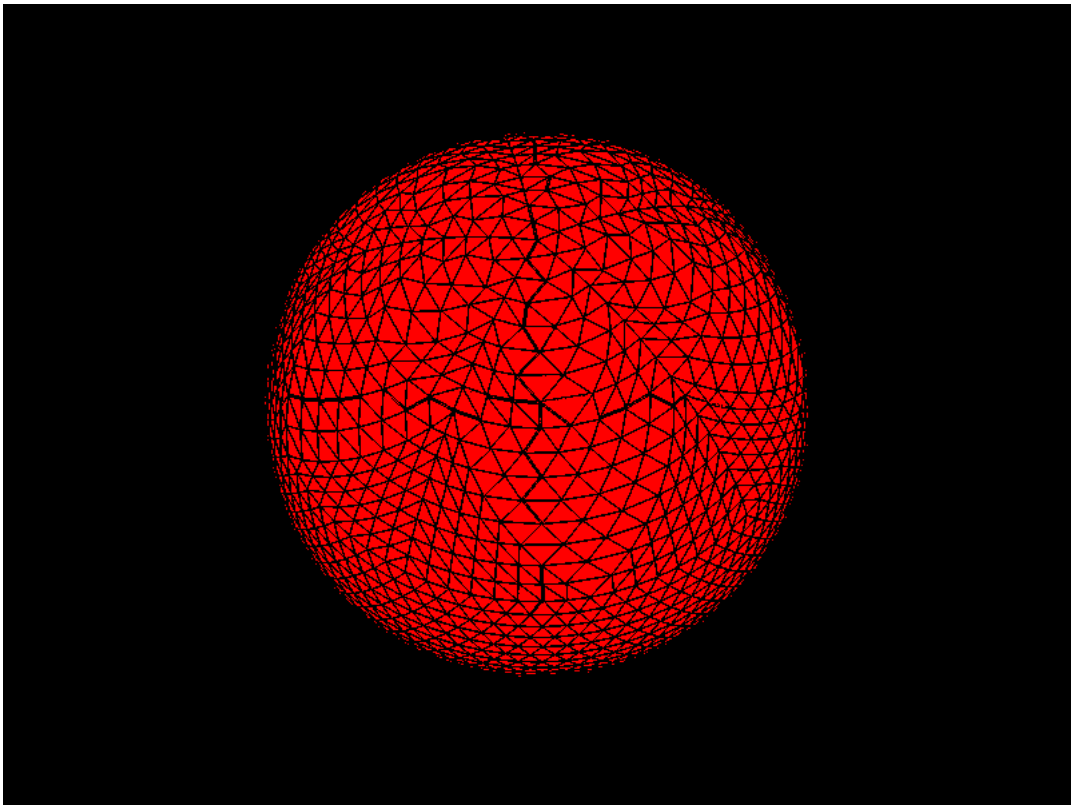

Figure 14: Type-3 subdivision of an icosahedron at level 4

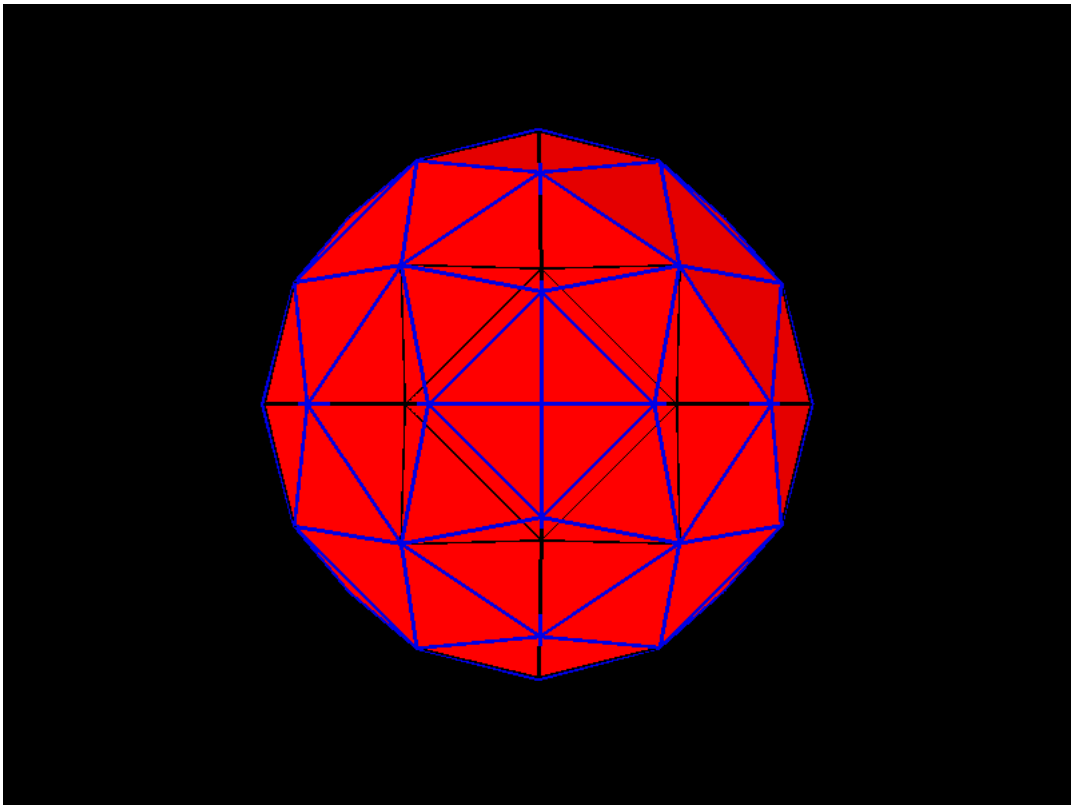

Figure 15: Type-1 and type-2 subdivisions of an octahedron at level 2

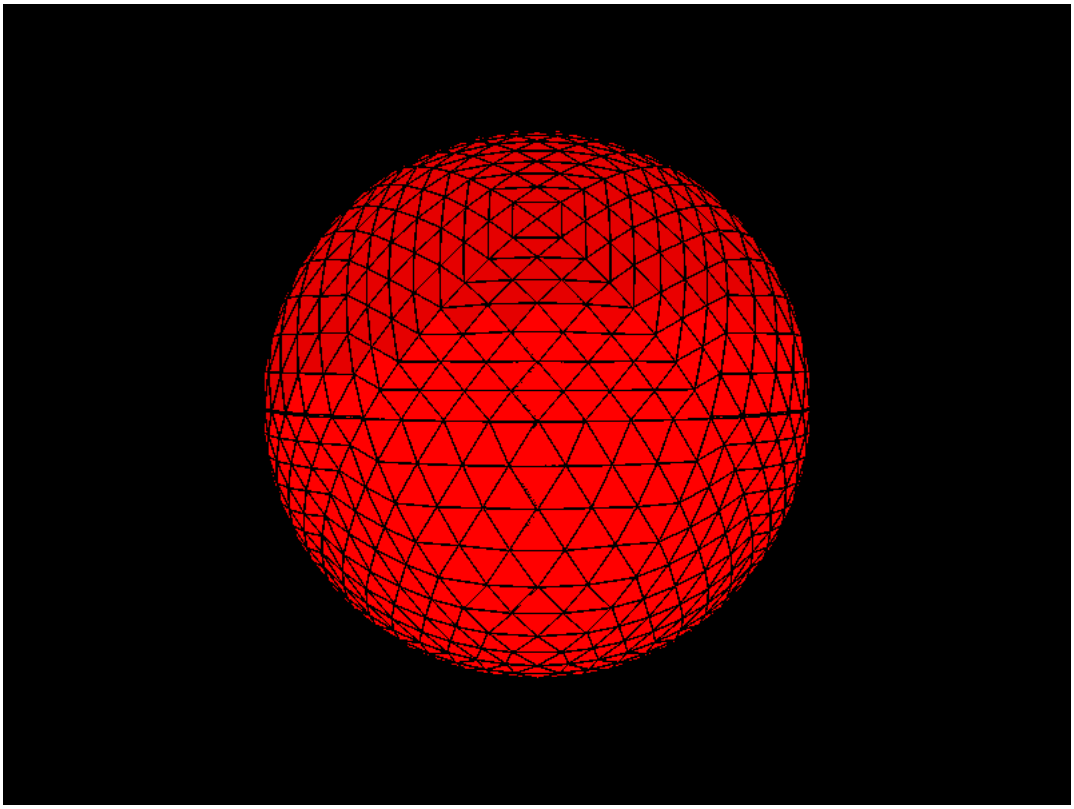

Figure 16: Type-2 subdivision of an octahedron at level 4

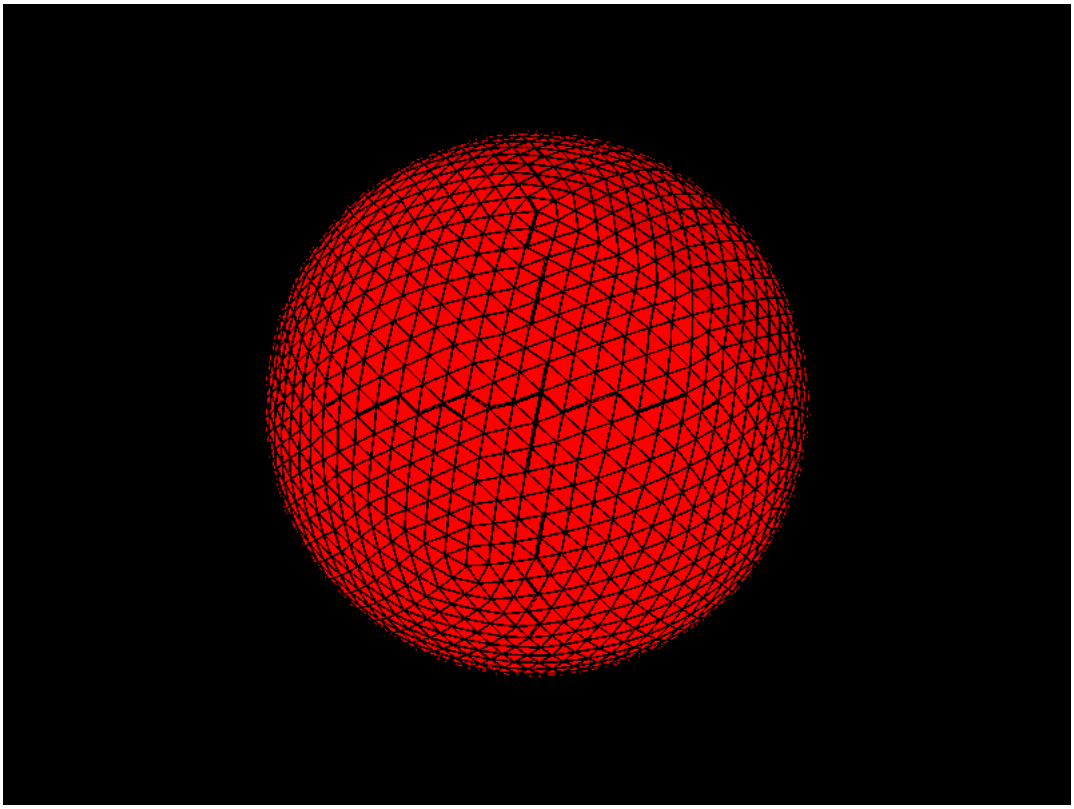

Figure 17: Type-2 subdivision of an icosahedron at level 4

References

[1] M. F. Goodchild and Y. Shirena, CVGIP: Graphical Models and Image Processing. **54, No1**, 31 (1990).

[2] G. Dutton, Cartographia **21**, 188 (1988).

[3] G. Fekete, in *Proc. Extracting Meaning From Complex Data: Processing, Display, Interaction* (PUBLISHER, Santa Clara, CA, 1990).