Geodesics on Curved 2D Surfaces Approximated by a Triangular Mesh

Seagods!
seagods@hotmail.com

April 6, 2011

1 Introduction

We want to propagate geodesics on some curved surface. It could be a mountain terrain, or some awkward shaped asteroid. Provided there is some algorithm that provides us with a triangular mesh approximation for the surface, the method we describe here will work. We show results for the algorithm working on a triangulated sphere.

In the programme "Sphere.cpp" documented in my Quadsphere article, we introduced a quad-tree based triangulation on a sphere. The basis of the programme is the same in the "Geo" Programmes here, except that we now allocate memory dynamically instead of just setting the number of triangles and nodes to some large number. It will be recalled that each triangle "knows" which triangle is on which edge.

We shall assume anyone reading this has access to the document describing the octahedral decomposition of a sphere into a triangular mesh. It is downloadable from the website where this document is published. The surface may not necessarily be a sphere. It could be any shape, so long as the surface is single valued when viewed from the origin. If it is not single valued, it may be possible to shift the origin to a position where it is. This will not be possible, for instance, if the surface is a torus. In other cases it may be possible. At any rate, in the "Sphere" program, the neighbour information is known automatically.

We shall exploit that property here. We can find some point in some triangle, and then "March off" in some direction in such a way that the path follows a "straightest possible line" with in the surface triangulation which we shall call T_{t1}. This kind of line is a surface *geodesic*.

In most textbooks introducing the subject, the reader is asked to imagine a bug or small animal travelling in the surface while being unaware that it is curved. This is generally introduced in Riemann Geometry, where the surface is smoothly curved.

The problem here is much easier than the Riemann geometry problem, because the surface T_{t1} is piecewise Euclidean. It's made up of a lot of "flat" bits which are connected in such a way that the overall geometry is not flat.

Consider a diamond shape, made of paper, being folded into two triangles as in Fig.1. The shortest distance between A and B is the same before and after the fold as any displacement of the point where the line crosses the folded edge reveals.

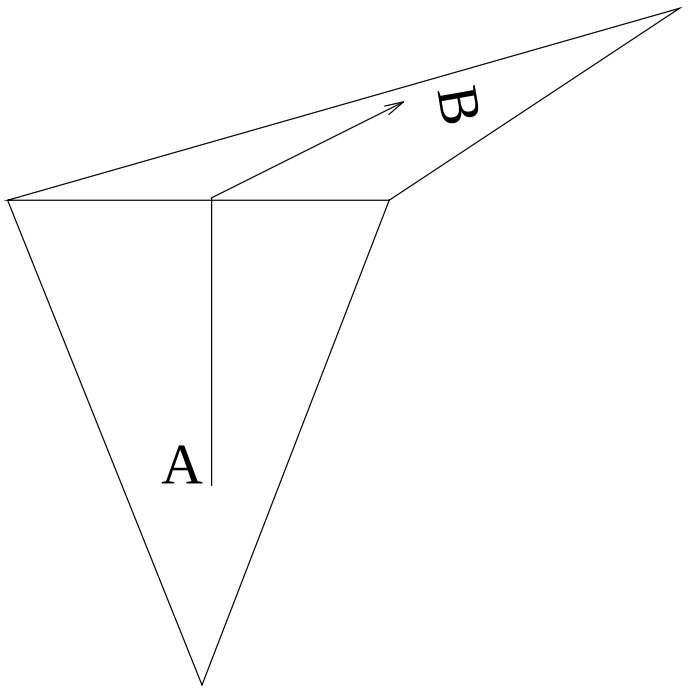

Figure 1: The shortest distance between two points in two neighbouring triangles.

As already stated, each triangle in T_{t1} must have a list of which triangles are at connected to its edges. Each triangle has its own coordinate system based on its edge vectors. Here we introduce another 2D coordinate system for each triangle which will be the usual 2S (x, y) coordinate system.

The origin shall be at the first node of the triangle, and the $\mathbf{i}$ unit vector is just $\mathbf{a}/|\mathbf{a}|$. Then we find the normal of the facet which is just $\mathbf{n} = \mathbf{a} \times \mathbf{b}$. Our y direction is then toward $\mathbf{n} \times \mathbf{a}$ so our $\mathbf{j}$ unit vector is a normalised version of this. All these systems have the same chirality because of the ordering of nodes 1, 2, and 3 is such that the nodes appear to run counterclockwise when viewed from outside of the surface. Any point within the triangle is then described by $\mathbf{r} = \alpha\mathbf{a} + \beta\mathbf{b}$. The edges of the triangle are on $\alpha = 0$, $\beta = 0$, and $\alpha + \beta = 1$. If α or β are greater than 1, or less than zero, the position is outside of the triangle, and if $\alpha + \beta > 1$ the position is also outside of the triangle.

When the ant reaches an edge and must travel further, it must find the triangle number and it needs to be in. It knows the neighbour, and it knows for instance that edge 2 of

the triangle it is leaving is edge 3 of in its neighbour on edge 1. Given this, it immediately knows the coordinates it has in the neighbouring triangle. If, for instance we are on edge on at some α and this happens to edge 1 in the next triangle, then $\alpha' = \alpha$. If the neighbours edge is in the opposite sense, then $\alpha' = 1 - \alpha$. It is always a simple matter to determine (α', β').

The more difficult problem it faces is this. If it has been following some direction vector $\mathbf{d} = d_1\mathbf{a} + d_2\mathbf{b}$, then what are the components (d_1', d_2') of the same direction vector in the second triangle? The second triangle shall have $\mathbf{i}'$ and $\mathbf{j}'$ as its Cartesian coordinate system's basis vectors. In general, these shall be rotated w.r.t each other. If we can find out the angle, we can find the components of $\mathbf{d}$ in the second system and continue on the journey.

The problem faced by the ant is depicted in Fig.2. Given α and β at the crossing point, we automatically know the values of α' and β' because we know which edge we are on in the next triangle. In the diagram, we land on edge 3 of the first triangle, which is the opposite of edge 3 in the next triangle.

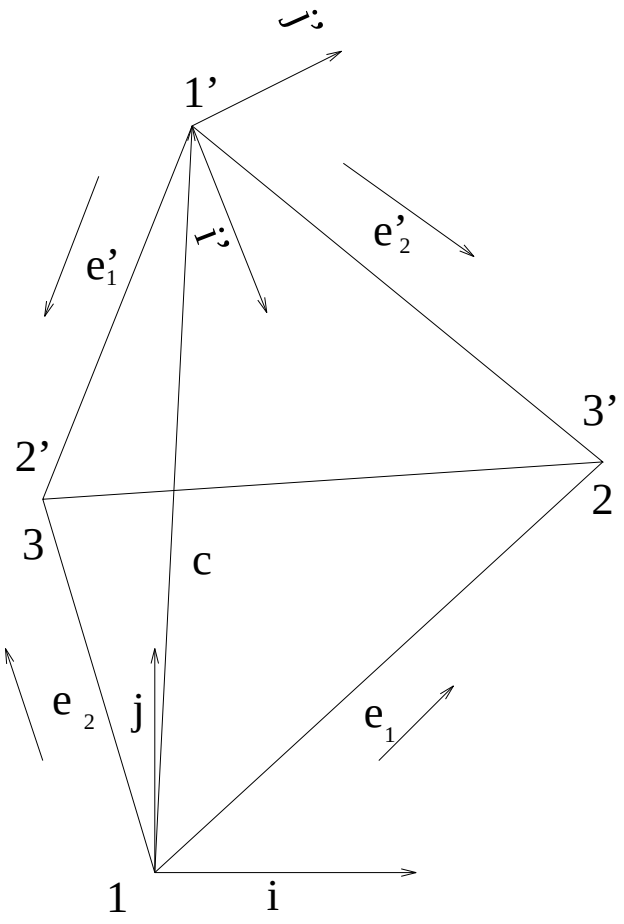

Figure 2: The two coordinate systems in the two triangles.

We also have a Cartesian system at the origins O and O' of each triangle (each at node 1). The thing is we do not have to use any 3D geometry. Flattening out the fold makes no difference at all to the local 2D coordinates. The lengths of the edge vectors and the angles between them are all unchanged. Because we have avoided any 3D calculations, we have no idea what the O' coordinates are w.r.t the first triangles coordinates, and we don't know what angle the primed 2D Cartesian coordinates are rotated through w.r.t the first triangle's 2D Cartesians. (In practise, we shall choose Cartesian coordinates such that $\mathbf{i}$ and $\mathbf{i}'$ are parallel to $\mathbf{a}$ and $\mathbf{a}'$, but we have drawn the more general case in Fig.2.) Also we do not know the translation vector $\overrightarrow{OO'}$ which we denote as $\mathbf{t}$.

So, how do we proceed? The following may look a little involved, but a toddler can do it. Suppose we have two paper triangles, and that some edge in one triangle has the same length as an edge in the other. All we are doing is spinning round one triangle and moving it so that the two edges match!

Our ant is on a shared edge which has nodes p and q. The triangle nodes shall be numbered 1 to 3 (as opposed to C-Programming style) so these edge nodes can be nodes 1 and 2 or 2 and 1, 2 and 3 or 3 and 2, or 1 and 3, or 3 and 1. The coordinates of the non Cartesian basis vectors $\mathbf{a}$ and $\mathbf{b}$ are written as (a_1, a_2) and (b_1, b_2) w.r.t. the local Cartesian system in triangle 1, and (a'_1, a'_2) and (b'_1, b'_2). The coordinates of the two edge nodes p and q are then

$$\begin{aligned}\mathbf{r}_p &= \alpha_p \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} + \beta_p \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \\ \mathbf{r}_q &= \alpha_q \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} + \beta_q \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}. \end{aligned} \tag{1}$$

Of course, since these are at triangle nodes, $(\alpha_{p/q}, \beta_{p/q})$ can only be one of $(0,0)$, $(1,0)$ or $(0,1)$ The same goes for these shared nodes in the primed coordinate system. We know that the primed Cartesians are rotated w.r.t the unprimed ones. That is

$$\begin{aligned}\mathbf{i}' &= \cos\Theta\mathbf{i} + \sin\Theta\mathbf{j} \\ \mathbf{j}' &= -\sin\Theta\mathbf{i} + \cos\Theta\mathbf{j}. \end{aligned} \tag{2}$$

We do not know the angle, but at the same time, we want to work in the two dimensions restricted to the surface. We put

$$\begin{aligned}\mathbf{a}'_{new} &= \begin{pmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{pmatrix} \begin{pmatrix} a'_1 \\ a'_2 \end{pmatrix} \\ \mathbf{b}'_{new} &= \begin{pmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{pmatrix} \begin{pmatrix} b'_1 \\ b'_2 \end{pmatrix} \end{aligned} \tag{3}$$

as the coordinates of $\mathbf{a}'$ and $\mathbf{b}'$ in the $\mathbf{i}$ $\mathbf{j}$ coordinates. We shall use the subscript *new* to denote the coordinates in the rotated system.

So we have

$$\alpha_p \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} + \beta_p \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} +$$
$$\alpha'_p \begin{pmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{pmatrix} \begin{pmatrix} a'_1 \\ a'_2 \end{pmatrix}_{new} + \beta'_p \begin{pmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{pmatrix} \begin{pmatrix} b'_1 \\ b'_2 \end{pmatrix}_{new}. \quad (4)$$

$$\alpha_q \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} + \beta_q \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} +$$
$$\alpha'_q \begin{pmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{pmatrix} \begin{pmatrix} a'_1 \\ a'_2 \end{pmatrix}_{new} + \beta'_q \begin{pmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{pmatrix} \begin{pmatrix} b'_1 \\ b'_2 \end{pmatrix}_{new}. \quad (5)$$

Here (t_1, t_2) are the components of the translation vector in the first triangle's Cartesian coordinates. We know all the everything but the angle Θ, so we can no solve for Θ.

We make things look simpler by putting

$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = \begin{pmatrix} \alpha'_p a'_1 + \beta'_p b'_1 \\ \alpha'_p a'_2 + \beta'_p b'_2 \end{pmatrix},$$
$$\begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} \alpha_p a_1 + \beta_p b_1 \\ \alpha_p a_2 + \beta_p b_2 \end{pmatrix},$$
$$\begin{pmatrix} W_1 \\ W_2 \end{pmatrix} = \begin{pmatrix} \alpha'_q a'_1 + \beta'_q b'_1 \\ \alpha'_q a'_2 + \beta'_q b'_2 \end{pmatrix},$$

and

$$\begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix} = \begin{pmatrix} \alpha_q a_1 + \beta_q b_1 \\ \alpha_q a_2 + \beta_q b_2 \end{pmatrix}, \quad (6)$$

then

$$\begin{pmatrix} \cos\Theta \\ \sin\Theta \end{pmatrix} = \frac{1}{((W_2 - X_2)^2 + (W_1 - X_1)^2)} \begin{pmatrix} (W_1 - X_1) & (W_2 - X_2) \\ -(W_2 - X_2) & (W_1 - X_1) \end{pmatrix} \begin{pmatrix} Z_1 - Y_1 \\ Z_2 - Y_2 \end{pmatrix}. \quad (7)$$

This determines the rotation part of the transformation. The translation vector is given by

$$\begin{pmatrix} t_1 \\ t_2 \end{pmatrix} = \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} - \begin{pmatrix} \cos\Theta & \sin\Theta \\ \sin\Theta & \cos\Theta \end{pmatrix} \begin{pmatrix} Z_1 - Y_1 \\ Z_2 - Y_2 \end{pmatrix}. \quad (8)$$

The dot products of the two vectors form the metric tensor for the triangle, i.e.

$$\begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{a}\cdot\mathbf{a} & \mathbf{a}\cdot\mathbf{b} \\ \mathbf{b}\cdot\mathbf{a} & \mathbf{b}\cdot\mathbf{b} \end{pmatrix} \quad (9)$$

We store the three independent coordinates of the metric at each triangle.

The ant's problem is that it must not change direction as it crosses the boundary. if it has been given a direction vector $\mathbf{d} = d_1\mathbf{a} + d_2\mathbf{b}$, then it must find its equivalent in the new coordinate system.

As we know the lengths of the local coordinates from $\sqrt{g_{11}}$ and $\sqrt{g_{22}}$ in the current and neighbouring triangles, and we know $\cos\Theta$, so we can immediately form quantities like

$$\mathbf{a}\cdot\mathbf{b}' = |\mathbf{a}|\mathbf{b}'|\cos\Theta. \quad (10)$$

If

$$\mathbf{a}' = x_1\mathbf{a} + y_1\mathbf{b}$$

and

$$\mathbf{b}' = x_2\mathbf{a} + y_2\mathbf{b} \tag{11}$$

then

$$\mathbf{d} = d_1\mathbf{a} + d_2\mathbf{b} = d_1'\mathbf{a}' + d_2'\mathbf{b}'. \tag{12}$$

So,

$$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} \begin{pmatrix} d_1' \\ d_2' \end{pmatrix}. \tag{13}$$

We can dot through eqn.11 with $\mathbf{a}$ and $\mathbf{b}$ to form

$$\begin{pmatrix} \mathbf{a}\cdot\mathbf{a}' \\ \mathbf{b}\cdot\mathbf{a}' \end{pmatrix} = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{a}\cdot\mathbf{b}' \\ \mathbf{b}\cdot\mathbf{b}' \end{pmatrix} = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}. \tag{14}$$

$$\begin{pmatrix} x_1 \\ y1 \end{pmatrix} = \frac{1}{g_{11}g_{22} - g_{12}g_{21}} \begin{pmatrix} g_{2,2} & -g_{12} \\ -g_{12} & g_{11} \end{pmatrix} \begin{pmatrix} \mathbf{a}\cdot\mathbf{a}' \\ \mathbf{b}\cdot\mathbf{a}' \end{pmatrix} \tag{15}$$

$$\begin{pmatrix} x_2 \\ y2 \end{pmatrix} = \frac{1}{g_{11}g_{22} - g_{12}g_{21}} \begin{pmatrix} g_{2,2} & -g_{12} \\ -g_{12} & g_{11} \end{pmatrix} \begin{pmatrix} \mathbf{a}\cdot\mathbf{b}' \\ \mathbf{b}\cdot\mathbf{b}' \end{pmatrix} \tag{16}$$

from which

$$\begin{pmatrix} d_1' \\ d_2' \end{pmatrix} = \frac{1}{x_1y_2 - x_2y_1} \begin{pmatrix} y_2 & -x_2 \\ -y_1 & x_1 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \tag{17}$$

The ant can now continue with no change in direction. This process is equivalent to ensuring that the first covariant derivative of the direction vector remains zero as the ant crawls along the surface, and the resulting curve is therefore a geodesic on T_{t1}.

The transformation of the components of the same vector $\mathbf{d}$ as it moves by parallel displacement will be familiar to the reader who has studied tensor calculus. However this is not what makes our space non-Euclidean. How can the 2D space as discussed here become non Euclidean? In Fig.3 we can see what is going on here.

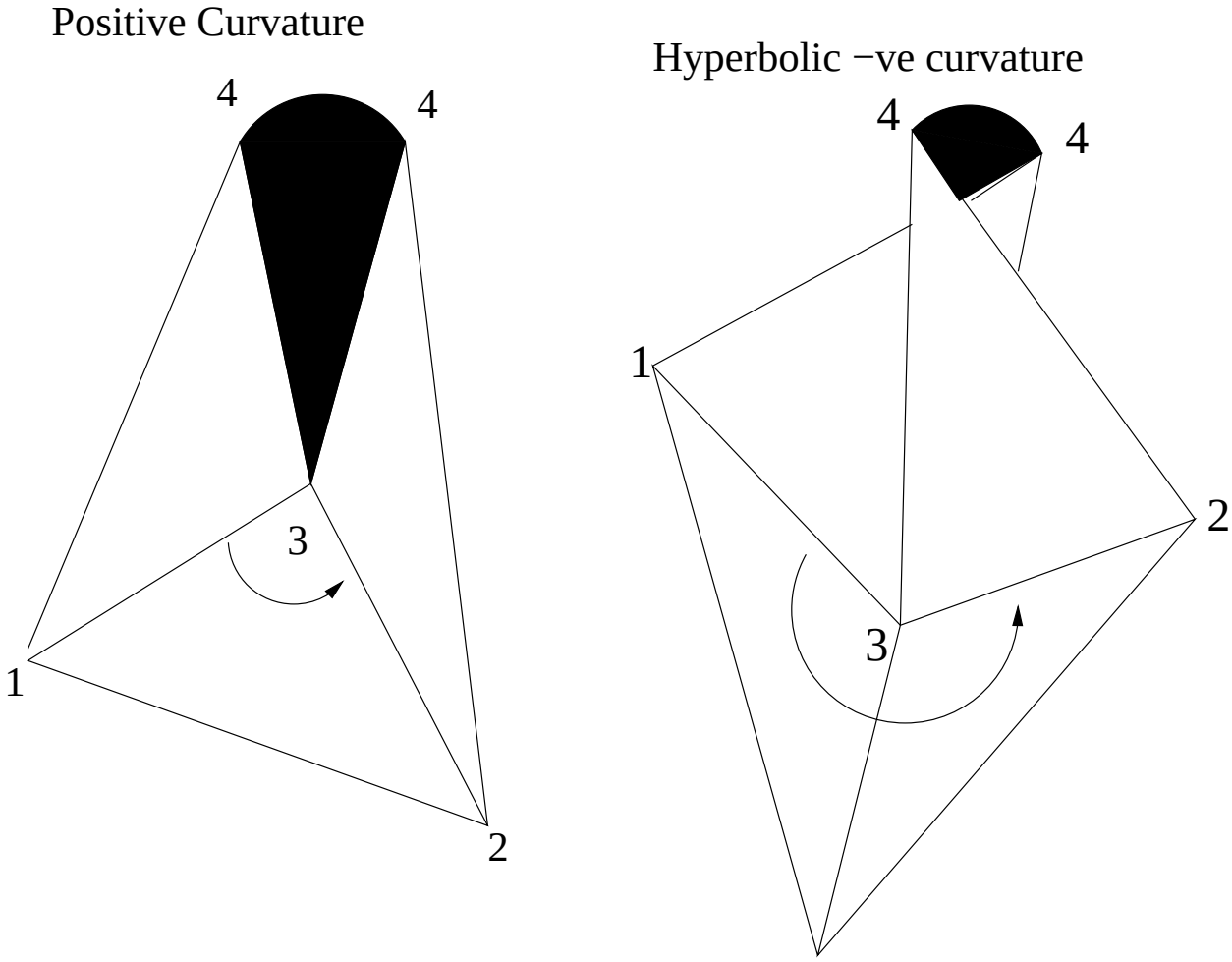

Figure 3: The "other node" in each neighbouring triangle is the same! The triangles are laid flat flat. On the left, we can fold them so that both nodes 4 are at the same point in 3D. The same can be done for the figure on the right where there is overlap rather than a gap.

What is happening is this. On the left we have a triangle with nodes 1, 2, and 3. It has neighbours on two edges. When "flattened out" there is a gap between the "other nodes" (both labelled none 4), but in fact they are the same node in the 3D space before we flatten the triangles out. To get back to the 3D shape, we have to fold along the edges and move the nodes numbered 4 out of the 2D plane which the triangle 123 lies until they meet again.

There is a fundamental difference between the two diagrams in Fig.3. If we flatten out all the triangles for a tetrahedron, icosahedron, or octahedron there are gaps between the edges in the 2D space. The same is true for the squares of the cube and the pentagons of the dodecahedron. These figures all have positive curvature.

In the diagram on the right, where we have an overlap rather than a gap, the curvature is *negative*. An arbitrary surface can have positive curvature, negative curvature, or some

places where the curvature is locally positive, and other places where it is locally negative. The signature of a locally negative curvature is a "saddle" shape.

The geometry of a space where the curvature is everywhere negative is called "hyperbolic". In a hyperbolic geometry, the angles of a triangle add up to less than 189°. On the sphere, the angles of a triangle consisting of two meridians 90° apart and the equator, add up to 270°. For spaces with positive (everywhere) curvature, the angles of a triangle add up to more than 180°. For the sphere, the curvature is equal to one, everywhere. There is a surface called a psuedosphere where the curvature is (almost) everywhere equal to minus one. There are singularities on this surface do we can't strictly say everywhere. It is the solid of revolution of a curve called a tractrix.

However if we take the hyperbola $y = 1/x$ and rotate that about the y axis in 3D to create a solid of revolution, we get a pair of surfaces where the curvature is negative everywhere. We shall leave off from topology with the remark that this property of whether we get gaps or overlaps when flattening out a surface (locally) tells us quite a bit!

To finish we make some remarks on the internal geometry of a triangle. Suppose we have some direction (d_1, d_2) vector which we wish to (actively) rotate through some angle ϕ to a new vector with coordinates $(d_{r1}, d + r2)$.

$$\begin{pmatrix} d_{r1} \\ d_{r2} \end{pmatrix} = \begin{pmatrix} \cos\phi + g_{12}\sin\phi/D & g_{22}\sin\phi/D \\ -g_{11}\sin\phi/D & \cos\phi - g_{21}\sin\phi/D \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}. \tag{18}$$

By active, we mean that the resulting vector is different from the original rather than the same vector but in coordinates of a rotated frame (which is usually called a passive transformation.

Suppose we are given some points $\mathbf{p}$ and $\mathbf{q}$ with coordinates (α_p, β_p) and (α_q, β_q). A straight line passing through $\mathbf{p}$ in the direction d is $\mathbf{p} + \lambda\mathbf{d}$. The square of the distance from the a point on the line is specified by λ. We want to find the point where the distance from $\mathbf{q}$ to the line is the closest possible. This is when

$$\frac{ds^2}{d\lambda} = g_{11}(2\alpha_p d_1 - 2\alpha_q d_1 + 2\lambda d_1^2)$$

$$+2g_{12}(\alpha_p d_2 + \beta_p d_1 - \beta_q d_1 - \alpha_q d_2)g_{22}(2\beta_p d_2 - 2\beta_q d_2 + 2\lambda d_2^2) = 0. \tag{19}$$

That is

$$\lambda = \frac{g_{11}(\alpha_q - \alpha_p)d_1 + g_{12}[(\beta_q - \beta_p)d_1 + (\alpha_q - \alpha_p)d_2] + g_{22}(\beta_q - \beta_p)}{g_{11}d_1^2 + 2g_{12}d_1d_2 + g_{22}d_2^2.} \tag{20}$$

In Fig4, our start point is at $\mathbf{s} = \alpha_s\mathbf{a} + \beta_s\mathbf{b}$. We march off in the direction $\mathbf{d}$ to some point $\mathbf{s} + \lambda\mathbf{d}$. We arrive on the edge $\alpha = 0$ when $\alpha_s + \lambda d_\alpha = 0$, the edge $\beta = 0$ when $\beta_s + \lambda d_\beta = 0$, and the edge $\alpha + \beta = 1$ when $\alpha_s + \lambda + \lambda d_\alpha = 1 - \alpha_s - \lambda d_\alpha$. So, we have three possible values of λ, namely $\lambda_1 = -\alpha_s/d_\alpha$, $\lambda_2 = -\beta_s/d_\beta$, and $\lambda_3 = (1 - \alpha_s - \beta_a)/(d_\alpha + d_\beta)$. Of course, negative λ values, or values which tell us the position on the edge is outside

the triangle are thrown out. If the starting point **s** is inside the triangle, we find only one sensible value, and that tells us which edge we arrive on.

In another article here we describe a quadtree subdivision In the Sphere.cpp programme, we have a triangulation where we know the neighbouring triangle numbers on each edge, and what the edge number and sense is in each of the neighbours as well. So, we know the initial position, and transform the components of **d** as above.

From now on, the above tests will give us two sensible results. Mostly, the one with a very small lambda can be thrown out, however we might land arbitrarily close to a corner and find two extremely small values of lambda. So, we need to keep an a track of whichever edge we started on and throw that option out. Otherwise a computer code may decide to put us back in the triangle from which we came, and we will be stuck on the edge forever.

As an illustration, the track in T_{t1} resulting from a repeated application of these rules is shown in Fig.4. It is clear that this is similar to a great circle on T, however most directions of travel do not result in a closed loop. The track will come back to a position close to the starting point, and the finer the triangulation T_{t1}, the closer the approach to the starting point. This shows that it is not build up of rounding errors that is the cause. The surface isn't quite a sphere.

2 Searching for the Shortest Path Between Two Points

Suppose we have two points P_s and P_t on the surface. We must remind ourselves that the surface is arbitrary up to the point that it can be represented by a triangular mesh. We are given the triangle number, and the (α, β) coordinates in each triangle. From the start point, we start with a given search direction. Next we try another, and we suppose each new search direction is to be at some set of angles to the first direction)

We can express the local Cartesian vectors in terms of $\mathbf{a}$ and $\mathbf{b}$. On putting $\chi = a_1 b_2 - a_2 b_1$ we see that

$$\begin{aligned} \mathbf{i} &= (b_2\mathbf{a} - a2\mathbf{b})/\chi \\ \mathbf{j} &= (b_1\mathbf{a} - a1\mathbf{b})/\chi \end{aligned} \tag{21}$$

Then given a direction $\mathbf{d} = \alpha\mathbf{a} + \beta\mathbf{b}$, then a rotated vector d' has coordinates

$$\begin{pmatrix} \alpha' \\ \beta\prime \end{pmatrix} = \begin{pmatrix} \cos\Theta & \sin\Theta \\ -\sin\Theta & \cos\Theta \end{pmatrix} (\alpha a_1 + \beta b_1)\mathbf{i} + (\alpha a_1 + \beta b_1)\mathbf{j} \tag{22}$$

Now we can use the expressions for the Cartesian unit vectors in terms of the triangle edge vectors. After some algebra we find that

$$\begin{pmatrix} \alpha' \\ \beta' \end{pmatrix} = \begin{pmatrix} \cos\Theta + \sin\Theta g_{12}/\chi & g_{22}\sin\Theta/\chi \\ g_{11}\sin\Theta/\chi & \cos\Theta - g21/\chi\sin\Theta \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix}. \tag{23}$$

So, given an angle to rotate the initial search direction, we immediately have the rotated direction in terms of the rotation angle and the metric tensor.

This is useful in our search for the target point. What if we arrive at the edge of a triangle at some point $\mathbf{p}$, and are travelling in a direction $\mathbf{dr} = d\alpha_r\mathbf{a} + d\beta_r\mathbf{b}$. We proceed along the line $\mathbf{r} = \mathbf{p} + \lambda\ bfdr$. We can use basic calculus to find that the closest approach is when

$$\lambda = \frac{g_{11}(\alpha_q - \alpha_p)d\alpha_r + g_{12}((\beta_q - \beta_p)d\alpha_r + (\alpha_q - \alpha_p)d\beta_r) + g_{22}(\beta_q - \beta_p)d\beta_r}{g_{11}d\alpha_r^2 + 2g_12d\alpha_r d\beta_r + g_{22}d\beta_r^2}. \tag{24}$$

Of course, now we know the coordinates of the point of closest approach, the relation $ds^2 = g_{ij}dx_i dx_j$ tells us how far away we get. If the reader is unfamiliar with the notation, a repeated index implies a summation, so in two dimensions the above is

$$ds^2 = g_{11}dx_1dx_2 + g_{12}dx_1dx_2 + g_{21}dx_2dx_1 + g_{22}dx_2dx_2. \tag{25}$$

In our case, the dx_1 the difference between the α coordinates of the point of closest approach and the target, and the dx_2 is the corresponding difference for the β coordinates.

3 Summary

We have demonstrated how to propagate a geodesic on an (almost) arbitrary surface. (For some surfaces, the triangulation may be impossible.) As well as this, we have outlined how

a search along a given set of angles at a uniform angle between the search directions can be made. We may for instance keep track of the 3D distance between the current triangle's centre and target triangles centre. Of course this will be zero if the search path hits the target triangle. The search may then be refined by keeping track of the the shortest distance of approach within the triangle, until we actually find the target point (within some preset tolerance.

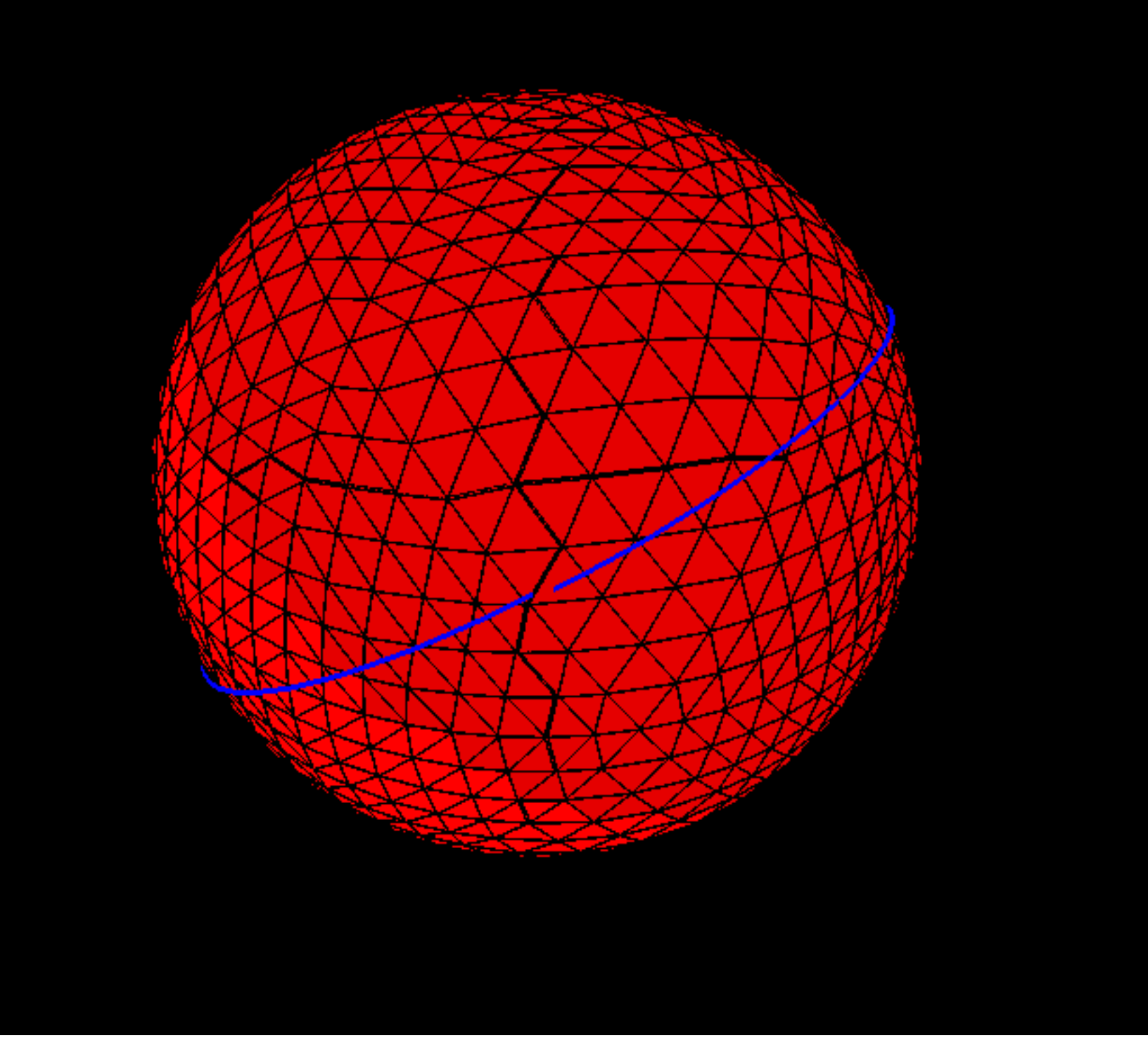

Figure 4: The process has gone round the sphere. The ends do not quite meet, because the surface isn't a sphere.