An exact algorithm for point location on spherical maps

Marcus Vinícius A. Andrade[1], Wagner F. Barros[1], Jorge Stolfi[2]

[1]Depto. Informática - Univ. Fed. Viçosa - MG - Brasil - 36570-000
{marcus,wbarros}@dpi.ufv.br
[2]Inst. Computação - UNICAMP - Caixa Postal 6176 - Campinas - SP - Brasil - 13083-970
stolfi@ic.unicamp.br

Abstract. We describe here an exact, and hence robust, algorithm for point location on *spherical maps* (maps on the sphere composed by arcs of circles, not necessarily geodesic ones). The algorithm relies on an exact representation for arcs of circles on the sphere, based on integer homogeneous coordinates, and exact geometric operations for such arcs. The topology of the map is represented by a specialized data structure (SMC — *spherical map by corners*) that supports fairly general map topologies, including closed edges without endpoints (*ovals*), isolated vertices, and faces with multiple borders. This framework makes it possible the construction of robust geographical information systems (GIS) and reliable algorithms for robotics, computer graphics, and other areas.

1 Introduction

A *spherical map* is a partition of the spherical surface into three kinds of *elements*: *vertices* (points), *edges* (circles and arcs of circles), and *faces* (open regions). Note that the arcs are not restricted to geodesics, but may belong to circles of arbitrary radius (i.e. to either "great" or "small" circles). See figure 1.

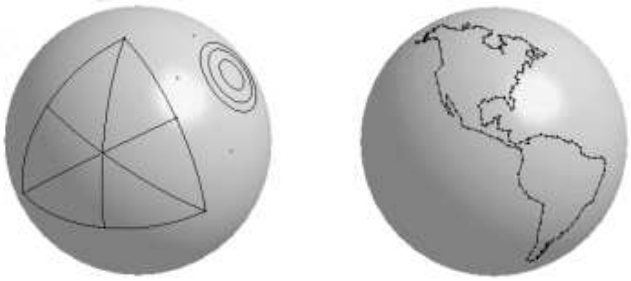


Figure 1: Examples of spherical maps.

This definition is general enough to cover most maps that arise in geographical information systems (GIS), including regions bounded by geodesic polygonal lines, flat polygonal maps under stereographic projection, latitude-longitude grids, footprints of aerial and satellite images, etc. [5]. Spherical maps also find applications in other areas such as robotics and computer graphics.

Here we briefly describe an exact representation for spherical maps [1, 2], and we give an exact algorithm for point location on such maps. The representation consists of two parts: a data structure that encodes the topology of the map, and an exact encoding of points, circles and circular arcs on the sphere.

The topological data structure, named SMC (*spherical maps by corners*), allows the representation of fairly general map topologies, including maps with oval edges (closed simple curves, not incident to any vertex), isolated vertices (not incident to any edge), and faces with multiple borders.

The geometry of the map is represented by various flavors of integer coordinate vectors, similar to the homogeneous coordinates of projective geometry. The representation provides dense sets of points and circles on the sphere, and allows us to define exact (rounding-free) geometric operations such as computing the points of intersection of two circles, the location of a point with respect to a circle, the ordering of circular arcs around a point, and the circular ordering of three points on a circle.

Exactness may seem a pointless luxury in GIS applications, since GIS data is by its nature approximate, and approximate results are sufficient for all practical purposes. However, most geometric algorithms used in GIS, such as point location and map overlay, become much more complex and prone to failure if their basic operations are subject to rounding errors, no matter how small [9]. Consider for example a distributed application that cuts a map into smaller submaps, handles each piece to a separate processor, and combines the partial results into a single map. If the cutting step is exact, the final step needs only to identify common boundary edges between the partial results, and remove them. The task becomes much harder if the cutting step is affected by rounding errors: the partial results may overlap, or may be separated by gaps. The

pasting operation is then almost impossible to specify, let alone to implement. Similar problems arise when we try to merge maps that cover overlapping regions but were developed under different flat projections.

The representation described here allows us to represent any spherical map with arbitrary accuracy, without increasing the number of elements of the original map; and also to develop exact (hence robust) algorithms for geometric operations on spherical maps, such as point location and map overlay [8]. The representation is such that the overlay of two maps can be overlaid again and again, without intermediate rounding or simplification steps, and without increasing the bit size of the coordinates.

All algorithms described here were implemented in C++ using the *GNU Multiple Precision* (GMP) arithmetic library [3] and are avaliable through the Internet[1].

2 Exact spherical geometry

We consider here only maps on the *unit sphere* $\mathbb{S}^2 = \{ (x,y,z) : x^2+y^2+z^2 = 1 \}$. The geometry of a map is then completely determined by the geometry of its vertices and edges, since these implicitly define the faces. We use here an exact representation for such elements which was proposed by Andrade and Stolfi [2].

Maps on some other spheroidal surface (such as the Earth) can be handled by establishing an appropriate correspondence between the latter and $\mathbb{S}^2$, e.g. by identifying points with same latitude and longitude. Even if this correspondence is not exactly computable, the approximation errors can be viewed as a mere input/output problem, similar to other data acquisition and display quantization errors, and do not affect the exactness of the internal representation nor the robustness of the geometric algorithms (which are all carried out on the $\mathbb{S}^2$ model sphere).

2.1 Points of $\mathbb{R}^3$

We represent a point $p = (X,Y,Z)$ of $\mathbb{R}^3$ by a vector of four *homogeneous coordinates* $[p_0,p_1,p_2,p_3]$, where p_0 is an arbitrary positive *weight*, $X = p_1/p_0$, $Y = p_2/p_0$, and $Z = p_3/p_0$. After Stolfi [7], we consider two homogeneous vectors to denote the same point if and only if they differ by a *positive* scalar factor. With this convention, the homogeneous vectors ($[0,0,0,0]$ excluded) comprise a double cover of projective three-space $\mathbb{P}^3$, called the *oriented projective space* $\mathbb{T}^3$. In particular, the homogeneous vectors with positive weight can be identified with the Cartesian space $\mathbb{R}^3$; and any vector of the form $[0,x,y,z]$ can be interpreted as a point at infinity in the direction of the Cartesian vector (x,y,z).

A point of $\mathbb{T}^3$ is said to be *rational* if it is a point of $\mathbb{R}^3$ with rational Cartesian coordinates, or the point at infinity along the direction of some rational vector of $\mathbb{R}^3$. It is easy to see that a point is rational if an only if it has an all-integer homogeneous coordinate vector.

2.2 Oriented planes

Dually, an *oriented plane* α of $\mathbb{T}^3$ is defined by a vector of four *homogeneous coefficients* $\langle \alpha_0,\alpha_1,\alpha_2,\alpha_3 \rangle$, except $\langle 0,0,0,0 \rangle$. The position of a point $p = [p_0,p_1,p_2,p_3]$ relative to α, is determined by $p \diamondsuit \alpha = \mathrm{sign}(\alpha_0 p_0 + \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3)$. By definition, the point p lies on α if $p \diamondsuit \alpha = 0$; otherwise p is said to be in the *positive* or *negative* half-space of α, depending on the sign of $p \diamondsuit \alpha$.

As in the case of points, two coefficient vectors denote the same plane if and only if they differ by a positive scalar factor. The *opposite* of a plane α is the plane $\neg\alpha$ that goes through the same points but has its positive and negative half-spaces swapped, namely $\langle -\alpha_0,-\alpha_1,-\alpha_2,-\alpha_3 \rangle$. The points at infinity are all incident to the *plane at infinity* $\Omega = \langle 1,0,0,0 \rangle$, and to its oppositely oriented version $\langle -1,0,0,0 \rangle$. A plane is said to be *rational* if it admits a coefficient vector whose entries are all rational numbers (or, equivalently, all integers). The *normal* of a plane α is the unit vector (direction) $\mathrm{nrm}(\alpha)$ that is parallel to the vector $(\alpha_1,\alpha_2,\alpha_3)$.

2.3 Oriented circles

We will use the term *S-circle* for an oriented circle (of nonzero radius) lying on the sphere $\mathbb{S}^2$. The orientation of an S-circle s consists of the choice of a *positive sense of travel* along s. We denote by $\neg s$ the same circle as s, but with the other orientation. The complement $\mathbb{S}^2 \setminus s$ of the S-circle consists two connected components, the *caps* or *sides* of s. The orientation of s and the right-hand rule allow us to distinguish the *positive side* from the *negative side*. By definition, the (*spherical*) *center* of the circle, denoted by $\mathrm{sctr}(s)$, is the point of $\mathbb{S}^2$ that is the center of its positive cap.

An *S-circle* s has a unique *supporting plane* $\alpha = \mathrm{pln}(s)$, which satisfies $s = \alpha \cap \mathbb{S}^2$ and is oriented so that the positive cap of s lies in the positive halfspace of α. Conversely, every oriented plane α that intercepts $\mathbb{S}^2$ defines a unique S-circle $s = \mathit{circ}(\alpha)$. See figure 2.
It follows that we can unambiguously represent an S-circle s by the homogeneous coefficients $\langle \alpha_0,\alpha_1,\alpha_2,\alpha_3 \rangle$ of the oriented plane $\alpha = \mathrm{pln}(s)$. We will write $((\alpha_0,\alpha_1,\alpha_2,\alpha_3))$ as a shorthand for

[1] `http://www.dpi.ufv.br/~marcus/smaps/`

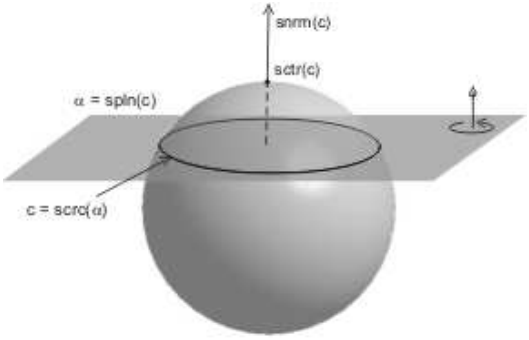


Figure 2: Example of an S-circle.

$circ(\langle \alpha_0, \alpha_1, \alpha_2, \alpha_3 \rangle)$. We say that an S-circle s is *rational* if $\mathrm{pln}(s)$ is a rational plane — in other words, if $s = ((\alpha_0, \alpha_1, \alpha_2, \alpha_3))$ where $\alpha_0, \alpha_1, \alpha_2, \alpha_3$ are integers.

A plane α intersects the sphere $\mathbb{S}^2$ at more than one point if and only if $\alpha_0^2 < \alpha_1^2 + \alpha_2^2 + \alpha_3^2$. When equality holds, the plane is tangent to $\mathbb{S}^2$, and the intersection reduces to a single point — which is *not* an S-circle, by our definition.

2.4 Rational and quasi-rational points

For exact spherical geometry, we need exact representations for points on the sphere $\mathbb{S}^2$ (S-points). An obvious class of points that admit such representation is the set $\mathcal{A}$ of the rational points of $\mathbb{R}^3$ that happen to lie on $\mathbb{S}^2$, such as $(2/3, 2/3, 1/3) = [3, 2, 2, 1]$. These are a proper subset of $\mathcal{B}$, the *quasi-rational* points, which are radial projections onto $\mathbb{S}^2$ of rational points of $\mathbb{R}^3$ — such as $(1, 2, 3)/\sqrt{14} = [\sqrt{14}, 1, 2, 3]$. More precisely,

$$\mathcal{A} = \left\{ [a_0, .. a_3] : a_0^2 = a_1^2 + a_2^2 + a_3^2,\ a \in \mathbb{Z}_*^4 \right\}$$
$$\mathcal{B} = \left\{ [b_0, .. b_3] : b_0^2 = b_1^2 + b_2^2 + b_3^2,\ b \in \mathbb{R} \times \mathbb{Z}_*^3 \right\}$$

where $\mathbb{Z}_*^k = \mathbb{Z}^k \setminus \{(0, .. 0)\}$.

Both $\mathcal{A}$ and $\mathcal{B}$ are dense subsets of $\mathbb{S}^2$. Note that the homogeneous coordinates of any S-point p satisfy $p_0 = \sqrt{p_1^2 + p_2^2 + p_3^2}$; so we can leave the weight p_0 implicit, and exactly represent any point of $\mathcal{B}$ (and hence of $\mathcal{A}$) by three integer coordinates p_1, p_2, p_3. Note also that $\mathrm{sctr}(s) \in \mathcal{B}$ for any rational circle s.

2.5 Sub-rational points

Geometric operations on spherical maps, such as map overlay, often need to compute the intersection points of two arbitrary rational S-circles. It turns out that those points may not be rational, or even quasi-rational. For instance, the S-circles $((1, 2, 2, 2))$ and $((1, 2, -2, 2))$ meet at the point $p = [4\ ,\ -1 + \sqrt{7}\ ,\ 0\ ,\ -1 - \sqrt{7}\ ]$ — which is not in $\mathcal{B}$, since the ratio $p_3/p_1 = (-1 - \sqrt{7})/(-1 + \sqrt{7})$ is not rational.

The intersection of two S-circles, in general, is either empty or consists of two S-points. To remove the ambiguity, we define *the* (*canonical*) *meeting point* of two S-circles s and t as being the point $s \wedge t$ where s is either tangent to t, or crosses t from its positive side into its negative side. See figure 3.

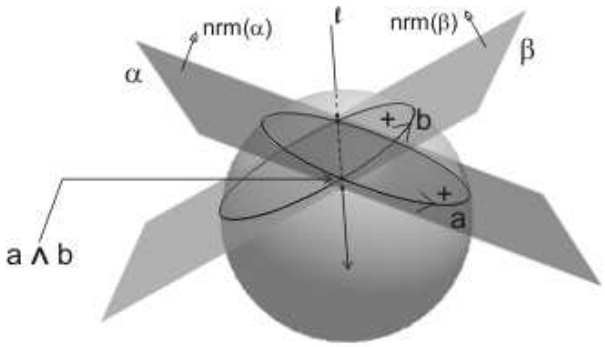


Figure 3: Intersection of two S-circles.

If the two circles s, t don't intersect, then, for the purposes of this paper, we define $s \wedge t$ and $t \wedge s$ as being a special value $\varnothing$, meaning "no point."

Notice that the set $s \cap t$ is equal to the intersection of the line $l = \mathrm{pln}(s) \cap \mathrm{pln}(t)$ and the sphere $\mathbb{S}^2$. For any two oriented planes α and β, we will denote by $\alpha \wedge \beta$ the line $\alpha \cap \beta$, oriented in the direction $\mathrm{nrm}(\alpha) \times \mathrm{nrm}(\beta)$— see figure 3. It turns out that the canonical intersection $s \wedge t$ of s and t is precisely the point $\mathrm{ext}(l)$ where the oriented line $l = \mathrm{pln}(s) \wedge \mathrm{pln}(t)$ exits the sphere $\mathbb{S}^2$.

The oriented line $\alpha \wedge \beta$ is uniquely determined by its six *Plücker coefficients* $\langle l_{01}, l_{02}, l_{12}, l_{03}, l_{13}, l_{23} \rangle$ where $l_{ij} = \alpha_i \beta_j - \alpha_j \beta_i$ [7]. For brevity, we will write $l = \langle l_0, l_1, l_2, l_3, l_4, l_5 \rangle$, where $l_0, .. l_5$ are the Plücker coefficients l_{ij} in the order shown above. Any six nonzero numbers define an oriented line, provided that they satisfy the *Plücker equation* $\pi(l) = l_{01} l_{23} - l_{02} l_{13} + l_{03} l_{12} = 0$; and two coefficient vectors define the same line if and only if they differ by a scalar factor. It follows that a *rational line* — a line defined by two rational planes, or two rational points — can be exactly represented by six integers $l_0, .. l_5$ satisfying the Plücker equation.

We denote by $\mathrm{ent}(l)$ the point where an oriented line l enters the sphere; and by $\mathrm{mid}(l)$ the midpoint of $\mathrm{ent}(l)$ and $\mathrm{ext}(l)$, which is also the point of l closest to the origin O of $\mathbb{R}^3$. Given $l = \langle l_0, .. l_5 \rangle$, let $\mu(l) =$

$l_2^2+l_4^2+l_5^2$ and $\delta(l) = \mu(l) - (l_0^2+l_1^2+l_3^2)$; we then have

$$\begin{aligned}
\text{ext}(l) &= [\,\mu(l),\ -l_1l_2 - l_3l_4 + l_5\sqrt{\delta(l)},\\
&\qquad\qquad l_0l_2 - l_3l_5 - l_4\sqrt{\delta(l)},\\
&\qquad\qquad l_0l_4 + l_1l_5 + l_2\sqrt{\delta(l)}\ ]\\
\text{mid}(l) &= [\,\mu(l),\ -l_1l_2 - l_3l_4,\ l_0l_2 - l_3l_5,\ l_0l_4 + l_1l_5]\\
\text{ent}(l) &= [\,\mu(l),\ -l_1l_2 - l_3l_4 - l_5\sqrt{\delta(l)},\\
&\qquad\qquad l_0l_2 - l_3l_5 + l_4\sqrt{\delta(l)},\\
&\qquad\qquad l_0l_4 + l_1l_5 - l_2\sqrt{\delta(l)}\ ]
\end{aligned}$$

From these formulas, we can see that the line l intersects the sphere (and the points $\text{ent}(l)$ and $\text{ext}(l)$ exist) if and only if $\delta(l) \geq 0$; and equality holds when l is tangent to the sphere. Therefore, the set of all intersection points of rational S-circles is precisely

$$\mathcal{C} = \{\, \langle l_0, .. l_5\rangle \,:\, \pi(l) = 0,\ \delta(l) \geq 0,\ l \in \mathbb{Z}_*^6 \,\}$$

which turns out to be a superset of $\mathcal{B}$. Further discussion of these formulas, and the relationship between $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ can be found in the paper by Andrade and Stolfi [2].

2.6 Geometric operations on S-circles

In general, the same point p of $\mathcal{C}$ may be the exit point of several rational lines. However, if p is in $\mathcal{C} \setminus \mathcal{A}$, it turns out that there is a *unique* rational line l such that $p = \text{ext}(l)$; whereas, if p is in $\mathcal{A}$, the oriented line l that joins the origin $O = [1,0,0,0]$ to p is rational and has that property. In either case, we call l the *canonical stabbing line* of p, and denote it by $\text{stab}(p)$. The integer Plücker coefficients of $\text{stab}(p)$, reduced of any common factors, provide then a canonical representation for the point p.

We will need also two predicates for testing the relative positions of S-circles and $\mathcal{C}$-points. We denote by $\otimes_s(p,q,r)$ the *relative circular order* of three points p, q and r on a S-circle s that passes through them, which may agree ($\otimes_s(p,q,r) = +1$) or disagree ($\otimes_s(p,q,r) = -1$) with the orientation of s; or may be indeterminate because two or more of the points coincide ($\otimes_s(p,q,r) = 0$).

Similarly, we denote by $\otimes_p(r,s,t)$ the *relative circular order* in which three S-circles r, s and t leave a common point p. This predicate can be defined as $\otimes_o(p_r,p_s,p_t)$, where $p_r = r \wedge o$, $p_s = s \wedge o$, $p_t = t \wedge o$, and o is a vanishingly small S-circle with $\text{sctr}(o) = p$. Note that if two of the circles have the same tangent direction at p, their circular order will depend on their relative radii. Exact algorithms for these predicates were given by Andrade and Stolfi [2].

2.7 Oriented arcs

If p and q are any two points on an S-circle s, we define *the oriented spherical arc* (*S-arc*) *of s from p to q*, denoted by $(p, \widehat{s}, q)$, as being the non-empty set of points encountered as we move from p to q on s, along its positive sense of travel. See figure 4.

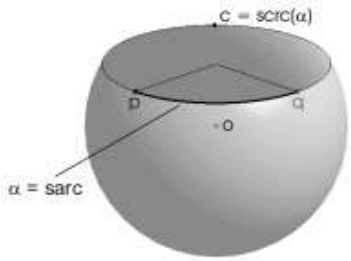


Figure 4: Example of an S-arc.

In particular, if p and q are the same point, there is only one S-arc on s, comprising the whole circle minus that point. It is convenient to allow also $p = q = \varnothing$, in which case the S-arc $(p, \widehat{s}, q)$ is, by definition, the whole circle s. It follows that an S-point $x \neq \varnothing$ belongs to that arc if $x \diamondsuit s = 0$, and either $p = q \neq x$, or $\otimes_s(p,x,q) = +1$.

Note that an S-arc is always non-empty, and has a definite orientation, namely that of the circle s. If $a = (p, \widehat{s}, q)$, we write $s = \text{circ}(a)$ (the *supporting circle* of a), $p = \text{org}(a)$ (the *origin*) and $q = \text{dst}(a)$ (the *destination*). We also denote by $\neg a$ the *reversal* of a, namely the arc $(q, \widehat{\neg s}, p)$ that contains the same points, oriented the opposite way. The arc is *proper* if $p \neq q$.

It is convenient to extend the S-circle intersection operator '$\wedge$' to S-arcs: namely $a \wedge b$ is defined to be the point $\text{circ}(a) \wedge \text{circ}(b)$, if that point belongs to a and b, or $\varnothing$ otherwise.

An S-arc is *rational* if has a rational circle and its endpoints are sub-rational (or $\varnothing$). Note that if a and b are two rational S-arcs, then $a \wedge b$ is either sub-rational or $\varnothing$. Note also that removing any sub-rational point from a rational arc breaks it into one or two rational arcs.

2.8 Rational maps

We propose to consider only *rational maps* on the sphere, whose vertices are sub-rational S-points, and whose edges are rational S-circles and S-arcs. This is not a significant restriction for practical purposes, since the set of rational arcs is dense in the set of all S-arcs, under the Hausdorff metric [2]. In other words, given an arbitrary S-arc a and a real $\epsilon > 0$, there is a ratio-

nal S-arc a^* such that the Hausdorff distance between a and a^* is at most ϵ.

3 Representing the topology

We represent the topology of a spherical map by the the SMC (*Spherical Maps by Corners*) data structure proposed by Andrade and Stolfi [2]. The SMC is a variant of the well-known *half-edge* data structure [6], extended so as to handle maps with isolated vertices, ovals (edges without endpoints), and faces with any number of borders.

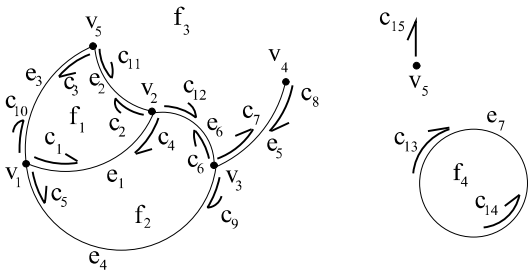
The chief concept of the SMC structure is that of *map corner*: a triple $\lfloor v, e, f \rfloor$, where, in general, v is a vertex, e is an oriented edge incident to v, and f is a face incident to v and e. We may think of a corner as identifying a position on the map: "at vertex v, facing towards the local direction of edge e, with the left foot on face f". Thus each undirected edge gives rise to precisely two distinct corners, $\lfloor v', e, f' \rfloor$ and $\lfloor v'', \neg e, f'' \rfloor$, where $\{v', v''\}$ are the endpoints of e, and $\{f', f''\}$ are the faces incident to e. Note that we may have $v' = v''$ or $f' = f''$.

In some cases, field v and/or field e may be $\varnothing$, denoting "no element." In particular, if e is an oval edge, its two corners have the form $\lfloor \varnothing, e, f' \rfloor$ and $\lfloor \varnothing, \neg e, f'' \rfloor$, and their implied position is "somewhere on e". In the same spirit, each isolated vertex v gives rise to a single corner $\lfloor v, \varnothing, f \rfloor$, where f is the only face incident to v. See figure 5.

The trivial spherical map — with only one face, no vertices and no edges — has no corners, by definition.

In the SMC, the adjacency relations between elements are given by three permutations of the corners, sym, onext and lnext. Given a corner $c = \lfloor v, e, f \rfloor$, sym($c$) returns the symmetric corner $\lfloor v', \neg e, f' \rfloor$, defined over the oppositely oriented edge $\neg e$. The function onext(c) returns the next corner after c around the vertex v, in the sense determned by a global orientation of the sphere. Finally, lnext(c) returns the next corner after c when walking along the frontier of f in the direction of the directed edge e. See figure 5.

Note that the same face may have multiple lnext orbits. Each of these orbits is called a *border* of f. The topology of a face f, and its connection to the rest of the map, is then determined by the (unordered) set of all its borders, denoted by frontier(f). With these four functions, one can "walk" from any corner to any other corner by a finite sequence steps.



corner	onext	lnext	sym
$c_1 = \lfloor v_1, e_1, f_1 \rfloor$	c_{10}	c_2	c_4
$c_2 = \lfloor v_2, e_2, f_1 \rfloor$	c_4	c_3	c_{11}
$c_3 = \lfloor v_5, e_3, f_1 \rfloor$	c_{11}	c_1	c_{10}
$c_4 = \lfloor v_2, \neg e_1, f_2 \rfloor$	c_{12}	c_5	c_1
$c_5 = \lfloor v_1, e_4, f_2 \rfloor$	c_1	c_6	c_9
$c_6 = \lfloor v_3, e_6, f_2 \rfloor$	c_9	c_4	c_{12}
$c_7 = \lfloor v_3, e_5, f_3 \rfloor$	c_6	c_8	c_8
$c_8 = \lfloor v_4, \neg e_5, f_3 \rfloor$	c_8	c_9	c_7
$c_9 = \lfloor v_3, \neg e_4, f_3 \rfloor$	c_7	c_{10}	c_5
$c_{10} = \lfloor v_1, \neg e_3, f_3 \rfloor$	c_5	c_{11}	c_3
$c_{11} = \lfloor v_5, \neg e_2, f_3 \rfloor$	c_3	c_{12}	c_2
$c_{12} = \lfloor v_2, \neg e_6, f_3 \rfloor$	c_2	c_7	c_6
$c_{13} = \lfloor \varnothing, e_7, f_3 \rfloor$	c_{14}	c_{13}	c_{14}
$c_{14} = \lfloor \varnothing, \neg e_7, f_4 \rfloor$	c_{13}	c_{14}	c_{13}
$c_{15} = \lfloor v_5, \varnothing, f_3 \rfloor$	c_{15}	c_{15}	c_{15}

Figure 5: A spherical map (flattened out on the plane), and its SMC representation.

3.1 Concrete representation

Concretely, the SMC data structure is implemented as a collection of records (objects) of four main types: Face, Vertex, Circle, and Corner.

Each Vertex record v stands for a vertex of the map, whose position pt(v) is a sub-rational S-point, defined by the Plücker coeffcients of its canonical stabbing line. Each Circle record contains the coefficients of a rational S-circle. As in the quad-edge data structure of Guibas and Stolfi [4], pointers to Circle records are always tagged with an orientation bit, so that the circles s and $\neg s$ can share the same record while retaining their respective orientations. Note that a single Circle record may be shared by two or more edges.

The edges are defined implicitly by the Corner records. A Corner record contains: pointers to the origin vertex org(c) (which is either a Vertex, or $\varnothing$) and to the left Face left(c); a tagged pointer to the underlying circle circ(c) (which is $\varnothing$ for corners of isolated vertices); and tagged pointers to the corners onext(c), oprev(c), and sym(c). Note that all other SMC corner functions can be derived from these, e.g. dst(c) =

org(sym(c)), lnext(c) = sym(oprev(sym(c))), etc.. In particular, the underlying edge is the S-arc edge(c) = $(p, \widehat{s}, q)$, where p = pt(org(c)), q = pt(dst(c)), and s = circ(c).

Finally, each Face object f points to a list frontier(f) of corners c with left(c) = f, one on each connected component of f's boundary. Presumably, clients will extend the Face, Vertex and Corner objects with other application-specific attributes of the elements.

The SMC structure for a spherical map $\mathcal{M}$ is usually handled by a pointer to one of its Face objects, denoted by root($\mathcal{M}$). Note that this convention works even for a trivial map, which has a single Face but no Vertex or Corner records.

4 Point location on a spherical map

The *spherical point location problem* is to determine which element (vertex, edge or face) of a given spherical map $\mathcal{M}$ contains a given point $p \in \mathcal{C}$.

We assume that the map is represented by an SMC structure, as described in section 3. For uniformity and usefulness, the procedure is required to return also a corner associated to the element in question, if there is one. More precisely, the algorithm must return an *address* for the point p, defined as a pair $l = (\mathrm{crn}(l), \dim(l))$ where $\dim(l)$ is the dimension (0, 1, or 2) of the map element containing p, and $\mathrm{crn}(l)$ is either a corner of that element, or $\varnothing$ if $\mathcal{M}$ is a trivial map. Note that a point p may have more than one address on a map. For example, if p lies on an edge e the result could be either $(c, 1)$ or $(\mathrm{sym}(c), 1)$, where c is a corner such that $e = \mathrm{edge}(c)$; and, if p coincides with a vertex v, then any pair $(c, 0)$ with $\mathrm{org}(c) = v$ would be adequate.

4.1 Main procedure: Locate

We now describe a procedure Locate($p, \mathcal{M}$) that solves the spherical point location problem, by computing an address for p on $\mathcal{M}$.

The procedure uses a simple *incremental walk* approach, often used for point location on plane maps. Namely, it simulates the motion of a point x along a *reference path* that starts at a point r, with a known address l_r, and ends at the given point p, updating the address l_x of x every time it crosses a vertex or an edge. When x reaches p, l_x will be the desired address.

The starting point r and its address l_r are easily generated. If $\mathcal{M}$ is the trivial map, then we pick r arbitrarily in $\mathcal{C}$, and set $l_r \leftarrow (\varnothing, 2)$. Otherwise, let $c = \lfloor v, e, f \rfloor$ be any corner of the map. If $v \neq \varnothing$, we set $r \leftarrow \mathrm{pt}(v)$, and $l_r \leftarrow (c, 0)$. Otherwise, e must be an oval edge, without endpoints; we pick any point r on the S-circle of e, and we set $l_r \leftarrow (c, 1)$.

Ideally, the reference path should be a single rational arc from r to p; however, such arc may not exist. (Two sub-rational points r and p lie on a common rational S-circle if and only if one of them is in $\mathcal{A}$, or the lines stab(r) and stab(p) are coplanar.) If that is the case for r and p, we generate a third point $u \in \mathcal{A}$, and build a reference path with two rational S-arcs $A_1 = (r, \widehat{s_1}, u)$ and $A_2 = (u, \widehat{s_2}, p)$, where $s_1 = \mathit{circ}(\mathrm{stab}(r) \vee u)$ and $s_2 = \mathit{circ}(u \vee \mathrm{stab}(p))$. Note that s_1 and s_2 are always rational circles.

In any case, the result is a path with at most two proper rational S-arcs connecting r to p. Locate then walks along each arc, in sequence, using the procedure LocateRel described below.

4.1.1 Single arc traversal: LocateRel

Given an arbitrary proper S-arc $a = (p, \widehat{s}, q)$ on the sphere, and an address l_p for its starting point p, the procedure LocateRel(a, l_p) returns the address l_q of its destination q. Like the global Locate procedure, LocateRel simulates the motion of a point x from p to q along the arc a, updating its address as it moves from one map element to the next.

In order to start this process, LocateRel must simulate an infinitesimal motion from the point p along the arc a, and compute an address $l_x = (c_x, d_x)$ for the displaced point x, given an address for p. This is not an entirely trivial task, since there are several cases to consider: p may belong to a vertex, edge, or face of the map, and the displaced point x may lie either somewhere along an edge, or inside a face. For this task, LocateRel uses the auxiliary procedure WhereTo(p, l_p, s), described in section 4.1.2.

Once the point x has started to move along the arc a, LocateRel must find the next significant event that may cause its address to change. If the point x is presently traveling along an edge e of the map (which must have circ(e) = circ(a) or circ(e) = $\neg$circ(a)), then the next significant event is either the end of e, or the end of a if it happens inside e. The procedure EdgeExit(a, c_x), described in section 4.1.3, is called to process this case.

If, on the other hand, the perturbed point x lies in the interior of a face f of the map, the next significant event is when x either reaches the frontier of f, or reaches the end of arc a while still inside f. The procedure FaceExit(a, c_x), described in section 4.1.4, is called to handle this case.

In either case, the outcome is an address $l_x = (c_x, d_x)$ for the point x where that event occurred. If x is the destination q of a, the LocateRel procedure is

finished, and l_x is an address for q. Otherwise, the procedure sets $\mathrm{org}(a) \leftarrow p \leftarrow x$ (thus discarding the part of a traversed so far), sets $l_p \leftarrow l_x$, and repeats everything from the beginning. Since the arc a may only intersect a finite number of edges and vertices of the map, this loop must eventually terminate with $x = q$.

4.1.2 Starting along an arc: WhereTo

We now describe the procedure WhereTo(p, l_p, s), which simulates an infinitesimal (arbitrarily small) advance of the point p, which has address $l_p = (c, d)$, along the S-circle s, in its positive direction; and returns an address l_x for the perturbed point x.

There are three main cases to consider, depending on whether the initial point p lies on a vertex, an edge, or a face of the map:

(i) If p lies inside a face f (that is, $d = 2$), then a sufficiently small displacement will remain inside f; return $l_x \leftarrow l_p$.

(ii) If p lies on some edge e ($d = 1$), the result depends on the position of s relative to the supporting S-circle $t = \mathrm{circ}(c)$ of e, at that point:

  (a) If $\otimes_p(t, s, \neg t) = +1$, then x immediately leaves the edge e and moves into the face $\mathrm{left}(c)$; return $l_x \leftarrow (c, 2)$.

  (b) Conversely, if $\otimes_p(t, s, \neg t) = -1$, then x immediately leaves e and moves into face $\mathrm{right}(c) = \mathrm{left}(\mathrm{sym}(c))$; return $l_x \leftarrow (\mathrm{sym}(c), 2)$.

  (c) Finally, if $\otimes_p(t, s, \neg t) = 0$, then either $t = s$ or $t = \neg s$, and any sufficiently small motion will leave x on e; return $l_x \leftarrow l_p$.

(iii) Finally, if p coincides with a vertex v of the map ($d = 0$), then WhereTo uses onext repeatedly to enumerate all the corners leaving v, until a corner c' is found that satisfies one of the following conditions:

  (a) $\mathrm{edge}(c') = \varnothing$. In this case, there are no edges incident to v, and therefore x starts moving into the face $\mathrm{left}(c')$.

  (b) $\mathrm{circ}(c') = s$. In this case, x starts moving along the edge $\mathrm{edge}(c')$.

  (c) $\mathrm{circ}(c') \neq s$ and $\mathrm{onext}(c') = c'$; that is, there is only one directed edge incident on p, and x is not moving along it. In this case x starts moving into the face $\mathrm{left}(c')$.

  (d) $\otimes_p(r, s, t) = +1$, where $r = \mathrm{circ}(c')$ and $t = \mathrm{circ}(\mathrm{onext}(c'))$. In this case too x starts moving into the face $\mathrm{left}(c')$, in the gap between two distinct consecutive edges.

If the enumeration stops with condition (b), return $l_x \leftarrow (c', 1)$; otherwise return $l_x \leftarrow (c', 2)$.

4.1.3 Walking along an edge: EdgeExit

The procedure EdgeExit(a, c) is called by LocateRel to simulate the motion of a point x along the proper S-arc $a = (p, \widehat{s}, q)$, which happens to start on and along an edge $e = \mathrm{edge}(c)$ of the map, until this situation changes — i.e., until x reaches an endpoint of either arc. The procedure is expected to return the final position of x, and an address l_x for that point. More precisely, if the motion stops at an endpoint v of e, then we must return v and an address $l_x = (c', 0)$ such that $\mathrm{org}(c') = v$; otherwise, we must return $x = q$, and an address on the edge e, say $(c, 1)$.

In principle, the procedure should determine the *first* endpoint that is encountered by x starting from $\mathrm{org}(a)$. However, the two arcs may overlap in such a way that, as x moves along a, it leaves arc e but reenters it later on. In this case, what happens between these two events is irrelevant for the purposes of point location. Thus, the EdgeExit procedure needs only consider whether $\mathrm{dst}(a)$ lies inside e. More precisely:

(i) If $\mathrm{dst}(a) \in e$, return $x \leftarrow \mathrm{dst}(a)$ and $l_x \leftarrow (c, 1)$.

(ii) Otherwise, the motion will end at an endpoint of e. If $\mathrm{circ}(c) = s$, set $c \leftarrow \mathrm{sym}(c)$. In any case, return $x \leftarrow \mathrm{pt}(\mathrm{org}(e))$, $l_x \leftarrow (c, 0)$.

4.1.4 Crossing a face: FaceExit

The function FaceExit(a, c) is called by LocateRel to simulate the motion of a point x along the proper arc $a = (p, \widehat{s}, q)$, which starts by moving into the face $f = \mathrm{left}(c)$. The procedure is expected to return the point x where this situation changes, and an address l_x for that point.

The situation may change either because x reaches the destination of a while still inside f, or because x reaches the frontier of f, at or before the end of a. In the first case, we return $x = \mathrm{dst}(a)$ and $l_x = (c', 2)$ where c' is some corner with $\mathrm{left}(c') = f$, possibly c itself. Otherwise, we return the point x where a hit the frontier, which may lie on a vertex or inside an edge of the map; and an appropriate address for it.

As in the case of EdgeExit, it may happen that the arc a leaves f only to re-enter it at some later

point — perhaps right away. In that case, those two events, and any others that may occur between them, are irrelevant for point location purposes; ans we might as well continue the simulation as if x had never left f.

Therefore, FaceExit actually returns the destination of a, if it lies inside f; otherwise, it returns only the *last* point where a exits f. See figure 6: in case (a) the S-arc a exits face f at points q_1, q_2, q_3, q_4 and q_5, so FaceExit returns the exit point q_5 with address $(c_2, 1)$. In (b), it returns the arc endpoint dst(a), for which $(c, 2)$ would be a valid address; and, in (c), it returns the vertex q, possbly with address $(c_1, 0)$.

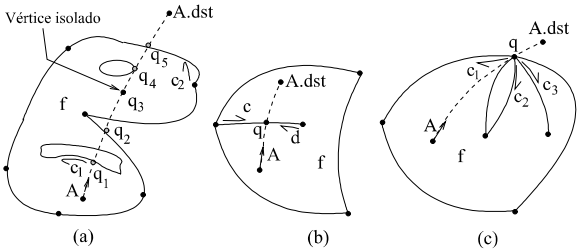


Figure 6: Traversing a face with FaceExit.

This seemingly minor optimization actually has a significant impact on the worst-case running time, as discussed in section 5.

Each point of intersection between the arc a and the frontier of f (which may comprise more than one connected component) must be either with a vertex $v = \text{org}(c')$ that lies on a, or with an (open) arc $e = \text{edge}(c')$ that has nonempty intersection with the arc a; where c' is a corner of f (that is, with $\text{left}(c') = f$). Note that an edge e of the frontier may contribute zero, one, or two intersection points; and may only touch the arc a, without actualy crossing it.

If the arc a goes through a vertex v, we can determine the status of the point x just after that event by looking at the relative position of the circle $s = \text{circ}(a)$ (and of its reverse) among all directed edges of the frontier of f that are incident to v. After leaving v, the point x will be inside f if v is an isolated vertex, or has at most a single incident edge e with $\text{circ}(e) \neq s$, or there are two distinct succesive corners c' and $c'' = \text{onext}(c')$, with $\text{org}(c') = \text{org}(c'') = v$ and $\text{left}(c') = f$, such that $\otimes_v(\text{circ}(c'), s, \text{circ}(c'')) = +1$.

When testing the above conditions for a vertex v, it is not necessary to enumerate its entire onext orbit (which may include many corners which are not incident to the face f). Instead, we use the fact that every corner c' that is relevant to the test is visited exactly once during the enumeration of the face's frontier. Thus we need to test those conditions only once for each visited corner c', at the time it comes up in the enumeration.

Finally, after we find an intersection point x, we may safely ignore any intersections that occur before it on a. Specifically, FaceExit(a, c) does the following:

(i) [*Initialize.*] Set $x \leftarrow \text{org}(a)$, $b \leftarrow a$, $q \leftarrow \text{dst}(a)$, $s \leftarrow \text{circ}(a)$, $l_x \leftarrow (c, 2)$. So, perform steps (ii) and (iii) for each corner $c' = \lfloor v, e, f \rfloor$ on the frontier of f:

(ii) [*Check for intersections with* $e = \text{edge}(c')$.]
  (a) If $e = \varnothing$, go to step (iii).
  (b) If $q \in e$, then terminate the FaceExit procedure, returning $x \leftarrow q$ and $l_x \leftarrow (c', 1)$.
  (c) Compute the intersections $x' = b \wedge e$, $x'' = e \wedge b$. If $x' = x'' = \varnothing$, go to step (iii).
  (d) If $x' = x''$, set $x \leftarrow x'$. Else, if $x' = \varnothing$, set $x \leftarrow x''$. Else, if $x'' = \varnothing$, set $x \leftarrow x'$. Else, set $x \leftarrow x''$ if $\otimes_s(x, x', x'') = +1$, $x \leftarrow x'$ otherwise.
  (e) Set $\text{org}(b) \leftarrow x$, $t \leftarrow \text{circ}(c')$. If $\otimes_x(t, s, \neg t) = +1$), then set $l_x \leftarrow (c', 2)$, else set $l_x \leftarrow (c', 1)$.

(iii) [*Check for intersection* $v = \text{org}(c')$.] If $v \neq \varnothing$, set $u \leftarrow \text{pt}(v)$, and do:
  (a) If $u = q$, then terminate the FaceExit procedure, returning $x \leftarrow u$ and $l_x \leftarrow (c', 0)$.
  (b) If u is inside the arc b, set $\text{org}(b) \leftarrow x \leftarrow u$, $l_x \leftarrow (c', 0)$.
  (c) If $u \neq x$, we are done with c'.
  (d) Otherwise, if $e = \varnothing$, set $l_x \leftarrow (c', 2)$;
  (e) Otherwise, let $t' = \text{circ}(c')$; if $t' = s$, set $l_x \leftarrow (c', 0)$;
  (f) Otherwise, let $c'' = \text{onext}(c')$; if $c'' = c'$, set $l_x \leftarrow (c', 2)$;
  (g) Otherwise, let $t'' \leftarrow \text{circ}(c'')$. If $\otimes_x(t', s, t'') = +1$, then set $l_x \leftarrow (c', 2)$.

(iv) [*Finalize.*] If the enumeration of frontier(f) ends without a premature exit from steps (ii)(b) or (iii)(a), then the remaining arc b, including its destination q, doesn't intersect said frontier. Then, if $\dim(l_x) = 2$, q is inside f, so faceExit returns $x \leftarrow q$ and l_x. Otherwise, $x = \text{org}(b)$ is the last point where u leaves f, so x and l_x are returned.

The key invariants of this algorithm, valid just before steps (ii) and (iii), are: (I) $b = (x, \widehat{s}, q)$ is a terminal piece of the original arc a. (II) No frontier element found so far contains q or intecepts b. (III) If $\dim(l_x) = 2$, then points infinitesimally ahead of x

(but not necessrily x itself) lie inside the face f, and l_x is a valid (face-style) address for those points. (IV) If $\dim(l_x) \leq 1$, then x lies on the frontier of f, and l_x is a valid (vertex-type or edge-type) adddress for x.

5 Complexity of the Locate algorithm

We define the complexity of a spherical map as being the number of corners in its SMC representation; and the complexity of an element is the number of corners that reference it. This is a useful metric because most other natural metrics (number of vertices, edges, faces, etc.) are bounded by the number of corners.

It is easy to see that the cost of WhereTo(p, l, s) is $O(1)$ if p lies on a face or an edge; and $\Theta(k)$,in the worst case, when p coincides with a vertex of complexity k. Furthermore, the worst-case cost of EdgeExit(a, c) is $O(1)$, and that of FaceExit(a, c) is $\Theta(k)$ where k is the complexity of the face left(c). It follows that the cost of LocateRel(a, l) is $\Theta(m)$, where m is the total complexity of all map elements that are intercepted by the S-arc a (or by its origin).

Note that the complexity of each face is counted only once in the measure m, no matter how many times it is entered by a. That is true only because FaceExit returns the last intersection with the frontier, rather than the first one. Without this optimization, the cost of LocateRel could be quadratic on the map complexity. Finally, since Locate performs at most two calls to LocateRel, we conclude that it too has worst-case complexity $\Theta(m)$, where m is the total complexity of all elements intercepted by the path — which, at worst, is equal to the complexity of the map.

6 Conclusions and future work

We have described here an original framework (apparently, the only one so far) for *exact* geometrical computations on spherical maps with general circular edges. The framework includes a data structure (SMC) for representing the topology of such maps; an exact representation for a dense set of points, circles, and circular arcs on the sphere; exact algorithms for basic geometric operations on such objects; and an exact algorithm for point location in those maps.

We believe that this framework can be quite valuable in many practical applications, such as robotics, computer graphics, and GIS. We note that a robust algorithm for computing the overlay two maps within this framework was given by Andrade [1]; and many geometrical operations in GIS can be reduced to map overlay and some trivial post-processing.

Our point location algorithm was presented mainly as proof of concept. Its large worst-case cost (linear on the number of traversed elements) restricts its use to small maps, or to applications where most queries are located close to each other. For random queries in large maps, one should pre-process the map into some efficient search structure.

Other promising themes for further research within this framework may be the development of space-efficient "multi-scale" representations of local geometric detail; good algorithms for approximating non-rational points and curves; and robust algorithms for other geometric operations, such as sweepline enumeration, topology-preserving rounding, and region expansion ("convolution").

Acknowledgements

This work has been partially supported by CNPq, Fapemig and Fapesp.

References

[1] M. V. A. Andrade. *Representação e manipulação exatas de mapas na esféricos.* PhD thesis, Instituto de Computação - UNICAMP, 1999.

[2] M. V. A. Andrade and J. Stolfi. Exact algorithms for circles on the sphere. *International Journal of Computational Geometry and Applications*, 11(3):267–290, 2001.

[3] T. Granlund. The GNU multiple precision arithmetic library. Technical report, Free software foundation, 1996.

[4] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[5] D. J. Maguire, M. F. Goodchild, and D. Rhind. *Geographical Information Systems - Principles and applications.* John Wiley & Sons, 1991.

[6] M. Mäntylä. *An Introduction to Solid Modeling.* Computer Science Press, 1988.

[7] J. Stolfi. *Oriented Projetive Geometry - A framework for geometric computations.* Academic Press, 1991.

[8] P. Y. F. Wu and Wm. R. Franklin. A logic programming approach to cartographic map overlay. *Canadian Computational Intelligence Journal*, 6(2):61–70, 1990.

[9] C. K. Yap. Towards exact geometric computation. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 405–419, 1993.