Rendering and managing spherical data with Sphere Quadtrees

György Fekete
SAR at National Space Science Data Center
NASA/Goddard Space Flight Center
Greenbelt, MD 20771

ABSTRACT

Most databases for spherically distributed data are not structured in a manner consistent with their geometry. As a result, such databases possess undesirable artifacts, including the introduction of "tears" in the data when they are mapped onto a flat file system. Furthermore, it is difficult to make queries about the topological relationship among the data components without performing real arithmetic. The sphere quadtree (SQT), which is based on the recursive subdivision of spherical triangles obtained by projecting the faces of an icosahedron onto a sphere, eliminates some of these problems. The SQT allows the representation of data at multiple levels and arbitrary resolution. Efficient search strategies can be implemented for the selection of data to be rendered or analyzed by a specific technique. Furthermore, sphere quadtrees offer significant potential for improving the accuracy and efficiency of spherical surface rendering algorithms as well as for spatial data management and geographic information systems. Most importantly, geometric and topological consistency with tha data is maintained.

1. INTRODUCTION

There are large collections of data with an inherent spherical distribution in the earth and space sciences. Traditional methods of managing data (e.g., flat sequential files for archived data, relational database management for metadata) are generally inefficient in storage or access for (spherical) spatial data. These limitations can severely constrain applications of large data sets like analysis and visualization. For example, earth scientists need to create two dimensional data visualizations such as images or contours, in an arbitrary geographic window and map projection ([11]). To provide such capabilities efficiently, including the drawing of overlays of world coastlines, requires that the data not be processed sequentially. A geometrically consistent, hierarchical representation of spherical data is discussed, which is a potential solution to such problems.

Generalizing the cartographic visualization idea implies a mechanism for drawing arbitrary maps quickly (see Figure 1) and to correlate geographic data of different resolutions. Disciplines such as planetary astronomy, solar physics and astrophysics also work with spherical data. Other applications of efficient access to spherical data include rapid search for data of interest in specific regions for analysis. Graphical browsing can be used to select areas tagged to specific data as metadata, which can then be displayed. For example, the browsing could use a world map as in Figure 1 for earth scientists or a star map for astronomers. In addition, construction of three dimensional models for the rendering of both volume and surface spherical data can benefit from a hierarchical representation.

Figure 1. An example of spherically distributed data

The representation called the *sphere quadtree* (SQT) described here and in [5] has many qualities desirable to accomplish the above tasks. For example, it is insensitive to the familiar distortions suffered in planar representations far away from the equator. Such distortions arise from the need to properly project data (i.e., flatten the earth) by preserving shape or distance, for example, in two dimensional visualizations. Problems relating to the relative location or adjacency of regions can be computed symbolically using a simple finite-state machine. Planar views of neighborhoods of up to about a quarter of the sphere around an arbitrary point on the sphere can be quickly generated for browsing purposes without performing any calculations involving conventional cartographic map projections. Another advantage of the SQT model is that it can be divorced from longitude–latitude based representations which have an inherent problem addressing regions in the immediate vicinity of the poles as a special case. In SQTs the poles are not distinguished, they are treated like any other datum point in the dataset. For example, switching between

CH2913-2/90/0000/0176/$01.00 – 1990 IEEE

a geographic and a geomagnetic polar reference frame poses no problems.

For visualizations purposes, we often reduce "continuous" datasets to a collection of colored polygons, which denote regions that represent some feature in the data. Connected components labeling (CCL) is an operation that groups adjacent elements with an invariant property. For example, given a gridded set of earth topography data and an invariant phrased as "elevation is above sea level", then CCL results in the assignement of unique labels to connected landmasses. CCL is not only useful for making pretty pictures, but also helps to characterize data. By exploiting the spatial structure in data managed in an SQT, CCL can extract information about the way space is fragmented by even sparse data. Consider, for example, a dataset consisting of only coastlines. There is no information explicitly present about the landmasses and the oceans, however, with CCL we can extract connected regions that represent landmasses or water. Figure 1 was generated this way. Geographic information systems (GIS) that maintain globally distributed data also benefit from using topologically consistent representation coupled with the ability to perform many operations definable by CCL. Connected components labeling is undoubtedly a very useful operation, hence the core algorithms for this paper were chosen as those necessary to support CCL.

2. BACKGROUND

Spherically distributed data are spatial data with a particularly inherent global structure that is closely related to a sphere. In the most general case there is a one-to-one correspondence between a position on a sphere, and a real number. We regard such associations as a function on the surface of the sphere called a *spherical image* (SI). For example, in a cartographical database, we would associate a position described by latitude and longitude with a number representing the label that best describes the position. The choice for labels depends on the application, but we may distinguish between two kinds of labels. The term *continuous spherical image* (CSI) refers to a functions whose range is continuous, such as elevation and temperature, for example. In contrast, *discrete spherical images* (DSI) describe spherical functions with discrete ranges, such as political or geological maps. In many cases DSI can be created from a CSI. Atlases that show elevation in a dozen colors or so are the result of coarse quantization of a continuous range of values. Regions of constant color are called *connected components*. Since for many applications making "maps" from either digital spherical images or quantized continuous spherical images is very important, the ability to perform connected components labeling on the surface of a sphere is very useful.

Region (or area) quadtrees are based on repeated decomposition of some finite, but arbitrarily large area into smaller regions. The resulting hierarchy is managed in a tree structure so that each non-terminal node has exactly four children. The structure aims to describe the whole region as a collection of piecewise simple regions. Thus large homogeneous regions yield shallow trees, whereas "busy" areas cause deep trees to be created. Planar quadtrees have been used extensively to model spatial data in the plane ([8,9]), but are not adequate in their original form to model spherical data, because planar quadtrees introduce increasing distortions as the plane is deformed to fit the spherical shell. We overcome this problem by modeling the sphere with a convex polyhedron, so that each face of the polyhedron is projected onto the surface of the sphere. The edges of the polyhedron become great circle segments that form the mesh which partitions the sphere. Polyhedra with more faces yield a finer mesh. Therefore, the solution for modeling the sphere is based on a recursive subdivision of the faces of a regular solid that results in a variable resolution of the underlying area.

The problem of finite approximation of the sphere by uniform patches is complicated by the fact that there are only five polyhedra, also called Platonic solids, whose sides are congruent simple equilateral polygons. These are the tetrahedron with 4 equilateral triangles for faces, the hexahedron (cube) with 6 squares, the octahedron with 8 triangles, the dodecahedron with 12 regular pentagons and the icosahedron with 20 triangles. Therefore any fine tesselation of the sphere is going to yield a number of different shapes, so the best we can hope for is a reasonable compromise between simplicity, regularity and multiple symmetry. Regularity requires that the variation in the shape and size of the patches be as small as possible. Multiple symmetry simply means that there are many rotations that do not change the appearance of the subdivided shape. For example, a cube-like subdivision has a sixfold symmetry. The simplest method of tessellating the sphere, often used by geographers, uses equally spaced meridians to divide the surface into wedges, which are in turn subdivided by lines of longitude. The intersection of the adjacent parallels with two lines of longitudes define a spherical quadrilateral. The greatest advantage of this scheme is the ease of computing the position and shape of each patch. However this method has many disadvantages. The symmetry is only twofold. Patches get smaller with increased distance from the equator, and in the neighborhood of the poles, the shape of the patches degenerates into triangles. These problems make this tessellation method less than regular. Projecting the edges of regular solids (polyhedra, whose faces are congruent simple polygons) onto the sphere gives us a better tessellation ([12,13]). Unfortunately, there are only five regular tessellations of the sphere, because there are only five regular solids. We have a slightly greater variety if we allow semi-regular polyhedra, which provide more faces, but consists of two kinds of basic shapes. Although the edges of the faces in this class of polyhedra are made up of segments of identical length, the difference in the areas of the two basic shapes can be large ([4]).

Finer approximation to the sphere can be obtained by subdividing the faces of the polyhedra recursively until the

desired resolution is reached. We chose the geodesic tessellation of the icosahedron ([4,6]) for our approximation to a sphere for the following reasons, which are also illustrated in Figure 2.

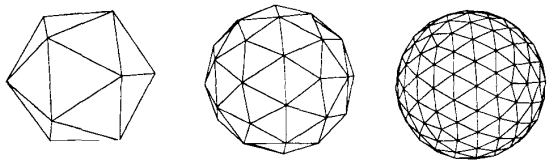

Figure 2. Icosahedron — subdivided once and twice

- Among the regular solids, the icosahedron has the most faces.
- Each face is a triangle.
- Subdividing any triangle produces exactly four new triangles.
- Subdivision involves only one kind of shape at all resolutions.

This regularity allows us to choose a well-known data structure, like the quadtree ([8,9]), to model a SI on a computer. The topic of refined regular solids for spherical modeling has received extensive treatment in [1] and [3].

On the CSI, for every position on the sphere with coordinates (ϑ, φ) there is assigned a value v, where ϑ and φ can denote longitude and co-latitude, for example. In principle, v may be a single number, or a vector of numbers, $\vec{v}$. The collection of 3-tuples (ϑ, φ, v) determines a surface around the sphere. One can think of this surface as a unit sphere perturbed by the v-values. For rendering purposes and in order to avoid denting the surface far enough to go through the center, the "unit" must be carefully chosen so that it is sufficiently larger than the smallest v. In the foregoing we shall assume that the surface is smooth, and therefore continuous.

We can approximate a CSI by fitting piecewise planar surface patches over small regions of the sphere. Such a surface can be characterized as a piecewise linear function of position. The position on the unit sphere in spherical coordinates is denoted by (ϑ, φ), and the surface of the CSI is written as $v = f(\vartheta, \varphi)$. One of the requirements that we impose on the SQT model is that the piecewise linear patches be determined in such a way that the modeled surface belonging to any one of them should be within a uniform error of the true values. This restriction may cause some areas of the sphere to be more densely fragmented into planar patches than others, depending on the behavior of the surface. Furthermore, adjacent patches must form a continuous surface.

3. SPHERE QUADTREES

SQTs approximate the sphere by successively subdividing the faces of the icosahedron and projecting new vertices onto the sphere. The initial coarse tiling has twenty triangles. This coarse tiling is refined by subdividing each triangle into four subparts by bisecting the edges of the icosahedral triangles and drawing the new vertices out to the surface of the sphere. The process can be repeated to gain any desired resolution. For example, Figure 3 shows all *trixels* that contain coastlines at 5 levels deep in a discrete sphere quadtree. We use the term *trixel* to stand for the area within a spherical triangle used in tiling the sphere (the word "trixel" in spherical images was originally intended to stand for the spherical triangular counterpart of "pixel" in planar images). The icosahedron (i.e., as shown in Figure 1) is unfolded to reveal all trixels in Figure 3.

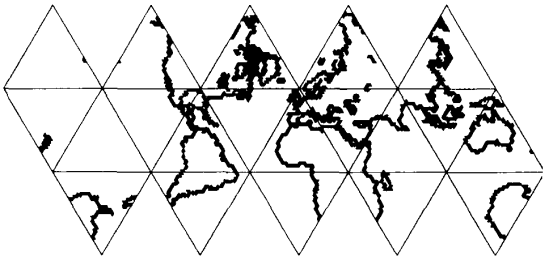

Figure 3. Coastline trixels of unfolded sphere quadtree

The SQT that represents the SI starts out as a forest of 20 quadtree nodes called *primitive trixels*, which represent the original triangles of the icosahedron . In Figure 4, two primitive trixels are shown resulting from projecting the shaded faces of the icosahedron onto the sphere. The four smaller trixels inside each larger one are the children of the primitive trixels.

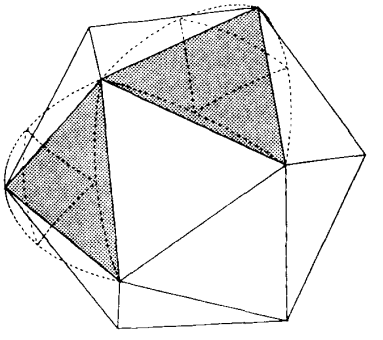

Figure 4. First four children of two primitive trixels

An SQT node models the behavior of a surface within the spherical shell represented by a trixel as a linear function of the position. In the spherical coordinate system φ is the angle between the $\mathbf{x}_3$ (or z) axis and the viewpoint, and ϑ is the angle between the $\mathbf{x}_1$ (or x) axis and the projection of the viewpoint onto the $\mathbf{x}_1\mathbf{x}_2$ (or xy) plane, as in Figure 5. So, for a trixel the linear surface patch is described by the four parameters t_1, t_2, t_3 and t_4 that satisfy

$$t_1\vartheta + t_2\varphi + t_3 v + t_4 = 0 \qquad (1)$$

for some value v, and values of ϑ and φ constrained to a region enclosed by the boundary of the trixel. Each node stores information about the data that are within the boundaries of the area described by the associated trixel. For continuous or gridded data, this information contains the parameters of the surface that best fits the data for the region. Several considerations make the triangle a very suitable geometric primitive. First, the parameters of a planar surface are uniquely determined by the position of three vertices and the associated values. Secondly, it is possible to completely tile the surface of a sphere with only triangles, although some may not be equilateral, depending

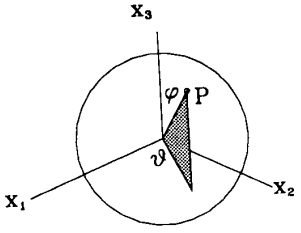

Figure 5. Position parameters ϑ and φ

on the size of the tiles. Moreover, recursively subdividing triangular regions into smaller triangular regions is a straightforward task.

The SQT is constructed incrementally as the trixels that need to be subdivided are identified. A trixel is split if the linear surface does not match the data surface well enough. This test is administered by sampling the data at four test points, and comparing the results with the corresponding point in the linear patch. The four testpoints are the centroids of the next generation of trixels. More precisely, if we let $rv(p)$ be the actual value of the SI at point p, and $ev(p)$ be the linear estimate from the same position, then we subdivide the trixel if

$$(rv(p) - ev(p)) \,/\, (rv(p) + ev(p))$$

is greater than some threshold value for any of the four testpoints, and the limit of the depth corresponding to the maximum desired resolution has not been met.

The subdivision of a trixel may cause the fragmentation of its neighbors. Suppose that trixel ABC in Figure 6 is subdivided, and that the new linear patches $A'BC'$, $A'B'C$ do not abut with DBC. Since this violates the requirement that adjacent patches must form a continuous surface, ABC must be subdivided to make the smaller patches abut with the patches inside DBC. This raises the question of whether or not this kind of forced division may propagate indefinitely, thereby destroying the advantage of the multiple resolution representation of the surface. Fortunately, the answer is no. Fragmentation caused by the splitting of a single trixel does not propagate beyond its immediate neighbors. Details of the discussion are in Appendix A.1.

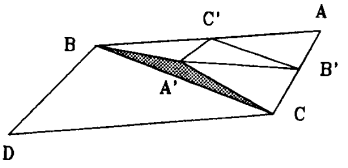

Figure 6. "Cracks" between adjacent trixels

4. TRIXEL GEOMETRY

Topological predicates, such as determining whether or not two regions are adjacent, normally involve computing using geometrical quantities. These calculations are expressed as computation with real numbers, which can be costly if done extensively. Therefore, it is very desirable to transform topological functions to simple symbolic operations whenever possible.

Our method for assigning names to trixels exploits the natural correspondence between trixels and nodes in the sphere quadtree. Topological predicates such as the test for adjacency between two trixels, and functions like finding the neighbors of a trixel are evaluated using an efficient, syntax-based operation on the name of the trixel that involves no floating point arithmetic. We introduce a scheme for labeling trixels, and present the algorithms for labeling connected components. on the sphere quadtree representation of a digital spherical image.

4.1 Definitions

Let (A, B, C) be the ordered list of vertices of a triangle representing a trixel. Let (A', B', C') be the set of midpoints of sides opposite A, B and C respectively, and let the label set consist of $\{1, 2, 3, 4\}$. We use symbols from the label set to mark each of the four triangles that can be constructed from the points (A, B, C) and (A', B', C') using rules as follows. Given the vertices of a trixel and the midpoints of the opposite sides, the children labeled "1", "2", "3" and "4" are matched, respectively, with the following triangles, and are illustrated in Figure 7. Each of the four ordered triples serves as the starting point (or the new A, B and C) for the next level, so the scheme can be applied repeatedly for a finer subdivision. The *name*

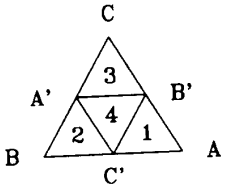

Figure 7. Children of a trixel and their labels

of the trixel is obtained by concatenating the node labels encountered on the path from the root. Figure 8 shows a root trixel subdivided twice with the names of all 64 trixels. The complete name of a trixel contains a reference to the root, which is one of the 20 primitive trixels. Hence the trixel names are of the form $\langle R_i, s \rangle$, to denote the path $\langle s \rangle$ starting from primitive trixel R_i. We sometimes use a short form for a trixel name like this, $\langle 1234 \rangle$, whenever the primitive trixel is understood or irrelevant. Two trixels that share an edge are said to be *adjacent*, or *neighbors*. For example, in Figure 7 $\langle 1 \rangle$ and $\langle 4 \rangle$ are adjacent but $\langle 1 \rangle$ and $\langle 2 \rangle$ are not.

When discussing neighbors of a trixel, it becomes necessary to identify relative *direction* between the adjacent trixels. Since trixels are triangular in shape, one way to define relative directions is by the segments joining the vertices of the triangle to the midpoints of the opposite sides. We label the three directions 1, 2 and 3. As an example, $\langle 1 \rangle$'s 1 neighbor is $\langle 4 \rangle$, and conversely, $\langle 4 \rangle$'s 1 neighbor is $\langle 1 \rangle$, as can be seen in Figure 7. Since triangles lack some of the symmetries enjoyed by squares, a trixel's neighbor designator is never ambiguous. Note that there is a one-to-one correspondence between the a vertex and a "1", "2" or "3" child of a trixel. This correspondence is used for defining names of vertices later. However, the notion of direction must be made more precise, because the sense of 1, 2 and 3 directions is not invariant, since the relative orientations of the trixels change at each subdivision. The root trixel will serve as the absolute reference frame for resolving all directions of any subtrixel level. We therefore need to track the changes in the direction labels for every descendant of the root. For example, inside $\langle 1 \rangle$ direction 1 is still 1. but directions 2 and 3 are switched when they are related to the root. The situation in $\langle 2 \rangle$ and $\langle 3 \rangle$ is similar. Relating the directions of further descendants to the root is achieved by recursively applying the substitutions of direction names for each level. Ultimately, the substitution can be represented as some permutation of the original three directions. The function **rd()** relates the direction relative to a given trixel to the root's absolute frame. The functions **neighbors()** and **rd()** are necessary to perform connected components labeling on an SQT. Vertices, or corners of trixels, belong to at least five and more often to six trixels (not counting vertices shared by a trixel and its descendants). It is desirable to store the results of any calculation associated with a vertex if it is needed later and the calculations are costly. The function **synonyms()** provides the means to find a *unique* name for a vertex to be used for reference.

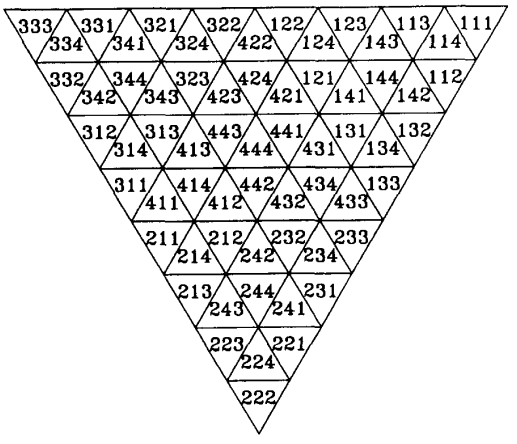

Figure 8. One primitive trixel subdivided twice

adjacent(s, t)	true if trixel s is adjacent to trixel t
rd(s, j)	direction represented by trixel s's local direction j
neighbors(s)	finds the 3 neighbors of trixel s
synonyms(s)	finds all valid names for a vertex

First, a restricted form of the algorithms that only operate within the confines of one icosahedral triangle is given to illustrate the principles behind the algorithms. The generalization to the entire sphere is a simple extension to this approach. The four fundamental properties of trixel geometry from which the algorithms are derived are as follows.

Property 1: Two trixels with common parents are adjacent if and only if one of them is a "4" trixel.
Property 2: Two adjacent trixels can take on only three different relative orientations.
Property 3: The adjacent subtrixels of two adjacent subtrixels are of the same type.
Property 4: If two trixels are adjacent, then their names differ in exactly one symbol (but not conversely).

Verification of these properties is left for Appendix A.3. The algorithm for the determination of adjacency refers to three configurations, so in order to easily name these we use three identifiers which are derived from the position of the dot at the vertex at "1" and imagining a circuit in the direction of "2" and then to "3", as in Figure 9. These identifiers are O (from "opposite"), S (from "shared") and C (from "counter-parallel").

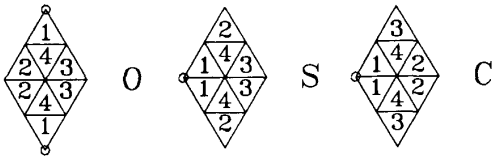

Figure 9. Three configurations for neighbor trixels

4.2 Connected components labeling

Connected components labeling, as the name implies, attempts to assign a unique label to each maximal connected region of trixels that share a common property. For example, a color coded map of the Earth's landmasses (for

simplicity's sake, surface above the mean sea-level) can be produced from Earth elevation data by grouping trixels into two bins according to the elevation value. Heights above sea level are in the *dry* bin, everything else is in the *wet* bin, which is also in the background. After applying the connected components labeling algorithm, a uniqe color is assigned to each *dry* component label before rendering. One bin must always be designated as background. The two-pass algorithm for SQTs is very similar to its conventional image processing analog ([7]). For determining connectedness, we must distinguish between are 3- and 12-neighbors. Two trixels are 3-neighbors if they share an edge, but if they have only one vertex in common, then they are called 12-neighbors (Figure 10). The reasons for the distinction is to maintain topological consistence in cutting up foreground and background. For example, in Figure 10 the single trixel wide vertical "stripe", which is 12-connected, ought to divide the background into two connected regions.

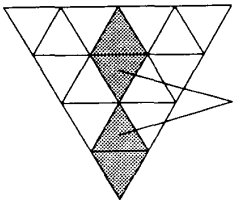

Figure 10.

If the background were also considered 12-connected, then it would remain as a single component, because some vertices are common to trixels on both sides of the stripe. Whether the background should be 3- or 12-connected is an arbitrary choice. Note, that there are trixels that contain vertices of the icosahedron that will have 11 instead of 12 neighbors, but fortunately this does not have an effect on the algorithm. The first pass of the algorithm assigns (not necessarily unique) labels to terminal nodes. The terminal nodes of the SQT are visited in some order, and the current node p is compared with its 3- or 12-neighbors that are in the same bin as p. Initially, all nodes are unlabeled. When a pair of adjacent nodes from the same bin, p and q_i are examined, then if neither has received a label before, then a new label is generated for both of them. If only one of them is labeled, then the unlabeled node gets the same label. If both nodes have already been labeled, then the two labels will have to be merged during the second pass. The function *equivalence* below enters this information into a table. The following program fragment summarizes the actions performed on the terminal nodes. The predicate L decides if its argument is labeled. The right arrow ($\rightarrow$) denotes that an action on the right side is performed when the condition on the left side is met.

foreach $q_i \in neighbors(p) \wedge bin(q_i) = bin(p)$ **do**

$\neg L(p) \wedge \neg L(q_i) \rightarrow label(p) := label(q_i) := newlabel()$

$\neg L(p) \wedge L(q_i) \rightarrow label(p) := label(q_i)$

$L(p) \wedge \neg L(q_i) \rightarrow label(q_i) := label(p)$

$L(p) \wedge L(q_i) \rightarrow equivalence(label(p), label(q_i))$

The second pass is responsible for sorting out the labels that have been recorded as equivalent. The result is an SQT in which a label identifier refers to a single component of constant value (bin). The example in Figure 11 shows the trixels in the coastline data set where there were only two bins to be considered: "coast" or "other". There was no information about land or water. The African continent was extracted after connected components labeling, and shading only trixels with the proper label. The correct label itself was extracted by peeking into a trixel known to be in Africa.

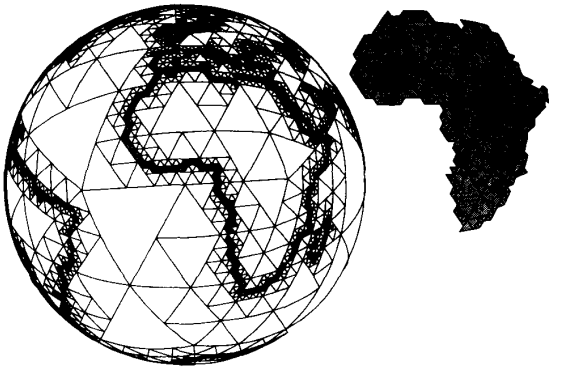

Figure 11. Coastline trixels, and connected component for Africa

4.3 Neighbor finding

Many trixel operations (connected components labeling, trixel chain code) involve finding the neighbors of a trixel, therefore the function that does this is perhaps the most important one of all. The algorithm presented here will find the 3-neighbors of a trixel. The 12-neighbor variation can be formulated in terms of this and the **synonyms()** functions. The algorithm first finds only those neighbors that are all within the same primitive trixel (original icosahedral triangle). In case the trixel is on a border, it also determines the relative direction, called the *open* direction of the border. In the context of this algorithm, a primitive trixel has no neighbors by definition, and all three sides are open. Neighbors in an open direction are found using other means, that turn out to be very simple. Often the direction of a particular neighbor, or the neighbor in a particular direction is of interest, therefore the neighbor finding function must also maintain global and relative directions of the adjacent trixels, and the open direction to the border whenever applicable.

In deriving the algorithm, the 4 fundamental properties of trixel geometry are very useful. If the trixel is a type "4", then the trixel is in the middle, therefore the neighbors are the three siblings. Otherwise, the trixel's has one type "4" sibling, and two other neighbors (not siblings) of the same type as the trixel. But the parents of these two neighbors are themselves neighbors. Therefore we may need to find out who the neighbors of the parents are, and by simi-

lar reasoning the neighbors of the grandparents, and so on all the way up to the root. This suggests that we ought to incrementally generate neighbors of the trixel in a top-down fashion. First, find the names of the neighbors of the root's first child, whose name consists of the first symbol in the original trixel's name, and then incrementally generate the names of possible neighbors for successively longer left substrings of the original trixel name. Thus, for a trixel $s_1 s_2 s_3 \ldots$ we first obtain the neighbors of s_1, then $s_1 s_2 s$, followed by $s_1 s_2 s_3$ until there are no more symbols to add.

At each step there are two cases to be considered: the partial trixel is either a type "4", or not. When it is type "4", then we already know that the three siblings are the neighbors. The neighbor in the relative directions 1, 2 and 3 is "1", "2" and "3", respectively. When the partial trixel is of another type, say "2", then its neighbor and sibling in the relative 2 direction a "4". The neighbors in the other two relative directions must be the same type as the current partial trixel. The type "4" neighbor has the same left substring up to the current symbol as the current partial trixel, and the other neighbors get the current symbol appended to them. Care must be taken to track the sense of the global directions with respect to the trixels indicated by the partial names. It is best to follow through this algorithm with an example. We shall generate the neighbors of ⟨1243⟩. The 3 rightmost entries in the table translate the partial trixel's local direction into the absolute reference frame. The details on how to do this is in the following section. The first line starts with the first partial trixel, ⟨1⟩, whose 1 neighbor is ⟨4⟩. The "·" under $\mathrm{nbr}_{2,3}$ means that there is a border in the indicated direction. For the next step, we know that the "4" neighbor is in the partial trixel's 2 direction. The "to-global" direction columns say that that local 2 is global 3, therefore the nbr_3 is now ⟨12⟩'s sibling, ⟨14⟩. For step 3, we use the direction columns to determine the global order of ⟨124⟩'s siblings. The first three steps are graphically depicted in Figure 12.

Nearest neighbors of ⟨1243⟩						
Partial trixel	nbr_1	nbr_2	nbr_3	to-global		
				1	2	3
⟨1⟩	⟨4⟩	·	·	1	3	2
⟨12⟩	⟨42⟩	·	⟨14⟩	2	3	1
⟨124⟩	⟨123⟩	⟨121⟩	⟨122⟩	2	3	1
⟨1243⟩	⟨1244⟩	⟨1213⟩	⟨1223⟩	3	2	1

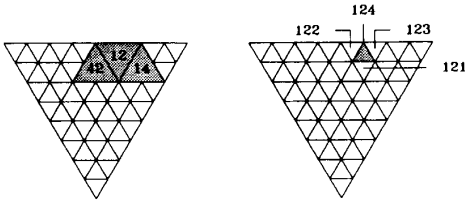

Figure 12. Generation of 3-neighbors of ⟨1243⟩

4.4 Relative directions

One of the advantages of the SQT representation is that it is not specific of any special coordinate system, and there are no designated special points (poles) or great circles (equator, ecliptic, galactic plane) to bias the choice of coordinate axes. But because there is no natural origin, a sense of direction will only be maintained within a primitive trixel. It is convenient to define a notation for the three directions using the symbols "1", "2" and "3". This notation is consistent with the position of the three children of the trixel in Figure 7. However, the orientation of subtrixels can be any one of six, the number of permutations of ("1", "2", "3"), The proper orientation of a trixel with respect to the root can therefore be identified by the appropriate row number from the above state table of permutations.

Permutation Table	
A	123
B	132
C	213
D	213
E	312
F	321

Relative direction state table						
	A	B	C	D	E	F
1	B	A	D	C	F	E
2	F	C	B	E	D	A
3	D	E	F	A	B	C
4	A	B	C	D	E	F

Finding the relative orientation of a trixel to its root is then accomplished by stepping through states representing permutations using the left-to-right scan of the trixel name and the above state table.

4.5 Adjacency

It is unfortunate, that the converse of Property 4 is not true. If it were, then determining whether or not two trixels are adjacent would be a simple matter of comparing two strings for an exact match within one symbol. However, it is possible to construct a finite-state acceptor that determines whether or not two strings represent names of adjacent trixels. For example, if we find that two adjacent trixels are in the O state, then their "1" labeled children are not adjacent. If their children are of type "2", for example, then these two nodes now take on a configuration corresponding to the C state, and they are adjacent. Let us augment the set of configurations with a forbidden state denoted by F. The rules for deciding which type of children are adjacent for any configuration is summarized in the following state transition table StatTable. The initial state is determined by the first non-identical pair of label symbols in the pathnames of the two nodes. InitTable defines the initial states.

InitTable				
	1	2	3	4
1	F	F	F	O
2	F	F	F	S
3	F	F	F	C
4	O	S	C	F

StatTable			
	O	S	C
1	F	C	S
2	C	F	O
3	S	O	F
4	F	F	F

The following algorithm implements the predicate that determines whether or not two nodes are adjacent. The inputs s_1 and s_2 are the names of nodes. The notation $s(k)$ signifies the k-th symbol in s.

Find smallest k such that $s_1(k) \neq s_2(k)$
CurrentState := InitTable($s_1(k), s_2(k)$)
while *CurrentState* $\neq F$ **do**
  begin $k := k + 1$;
  if $k >$ *length*(s_1) **then return True**;
  else return False;
  CurrentState := StatTable(*CurrentState*, $s_1(k)$);
  end;
return False;

It is possible in an incomplete quad-tree to have two adjacent nodes whose pathnames are not the same length. The shorter pathname denotes a larger node (that represents a larger area). If the two nodes are adjacent, then there must be an extension to the large node's pathname so that the extended name represents an adjacent node of the same size. Let the large and small nodes be represented by the names *head*$_1$..*tail* and *head*$_2$, respectively, so that *head*$_1$ and *head*$_2$ are the same length. Then there must be a string of symbols *extend* such that the pathnames *head*$_1$..*tail* and *head*$_2$..*extend* denote adjacent nodes. In the discussion of the adjacency predicate, it was pointed out that two adjacent nodes' pathnames must differ in exactly one symbol. So *head*$_1$..*tail* and *head*$_2$..*extend* must also differ in one symbol. But all ancestors of adjacent nodes are also adjacent, therefore *head*$_1$ and *head*$_2$ must be adjacent. So, it follows that the only different symbol must be in *head*$_1$ and *head*$_2$, and therefore *tail* must equal *extend*. Hence, the algorithm for testing adjacency can be easily modified for pathnames of different lengths by simply taking the tail of the longer pathname and appending it to the shorter one before applying the original algorithm.

4.6 Unique names for vertices

On a completely subdivided sphere all but 12 vertices are parts of six triangles (the exceptions are the vertices of the icosahedron). It would be unwise to recompute either the position or some parameters that are associated with these points as we encounter trixels randomly. Therefore, we need the ability to describe any vertex symbolically. The following observations help to derive a naming convention for vertices. Every trixel has three vertices which are contained in three subtrixels (called "outer" subtrixels) labeled 1, 2 and 3. Furthermore, every vertex is a part of exactly one of the outer subtrixels. Therefore, we may identify a vertex by specifying the trixel, and the outer subtrixel containing the vertex. Vertex names are of the form $\langle R_i,\ \mathbf{sx}\rangle$, where $\langle R_i,\ \mathbf{s}\rangle$ is the label of a trixel, and $\mathbf{x}$ is one of $\{1, 2, 3\}$. Although vertex names so constructed denote a single vertex unambiguously, each may have up to six different names since each vertex is shared by five or six trixels. Fortunately, finding unique names for vertices is easy because of the natural lexicographic ordering of names. To find a unique name for any given vertex, we adopt the convention of selecting the one with the lowest lexicographic value among the five or six synonyms. Generating the synonyms of a particular vertex V is analogous to generating all strings representing trixels which contain the vertex. By definition, the trixel T, whose name consists of the same string as V contains V. So, the task of generating all the synonyms for V is reduced to successively finding neighbors of T containing V, until all 6 (or 5) trixels have been encountered. In the below algorithm the function **neighbors()** finds the three neighbors of a trixel. What remains to be done is the elimination of the odd neighbor of T which does not contain V. It is evident from trixel geometry that the odd neighbor is the sibling whose name ends with **4**. The following algorithm constructs the set of all synonyms for any given vertex.

$S := \{T\}$
repeat until S does not change
foreach $T_i \in S$ **do**
  $S := S \cup \{\text{neighbors}(T_i) | \text{tailend}(T_i) \neq \mathbf{4}\}$
end

In Figure 13, for example, all the synonyms of vertex $\langle R_1,\ 1\rangle$ are found by first finding the two neighbors which do not end with a **4**, that is, $\langle R_6,\ 1\rangle$ and $\langle R_5,\ 3\rangle$. These in turn are added to S, and the process is repeated. The trace of the algorithm until completion is summarized in the table below. The column labeled *"add to S"*, contains previously unencountered names, which do not end with a **4**, from the *"neighbors"* group

add to S	neighbors
$\langle R_1,\ 1\rangle$	$\langle R_1,\ 4\rangle\ \langle R_6,\ 1\rangle\ \langle R_5,\ 3\rangle$
$\langle R_6,\ 1\rangle$	$\langle R_6,\ 4\rangle\ \langle R_{15},\ 1\rangle\ \langle R_1,\ 1\rangle$
$\langle R_5,\ 3\rangle$	$\langle R_5,\ 4\rangle\ \langle R_1,\ 1\rangle\ \langle R_{14},\ 2\rangle$
$\langle R_{15},\ 1\rangle$	$\langle R_{15},\ 4\rangle\ \langle R_{14},\ 2\rangle\ \langle R_6,\ 1\rangle$
$\langle R_{14},\ 2\rangle$	$\langle R_{15},\ 1\rangle\ \langle R_5,\ 3\rangle\ \langle R_{14},\ 4\rangle$

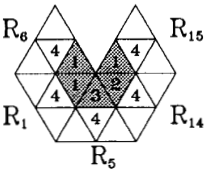

Figure 13. Synonyms of vertex $\langle R_1,\ 1\rangle$

To decide whether or not two vertex names refer to the same vertex, we generate the *unique* name for both, and test them for equality.

4.7 Extensions to the whole sphere

So far all algorithms work only within the boundaries of a primitive trixel. With the observation that adjacent trixels across primitive boundaries have isomorphic pathnames, the algorithms can be easily augmented to handle the entire surface of the sphere. For example, the two adjacent root trixels in Figure 14 the following symbol substi-

tution will cause names of trixels on one side of the border become the same name as their neighbor on the other side.

$$1 \rightarrow 3,\ 2 \rightarrow 2,\ 3 \rightarrow 1 \qquad (3)$$

The act of adding any number of levels (subdividing any number of times) does not call for a different substitution rule. This is very convenient, because only one substitution table need be stored for each pairing of primitive trixels. The neighbor finding algorithm would find the 3-neighbors of border trixels by simply translating the symbols in the trixel's name by using the appropriate substitution rule.

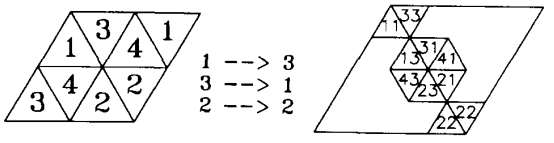

Figure 14. Association of trixel names within two adjacent primitive trixels

5. FUTURE DIRECTIONS

To gain further insight regarding the value of SQT representations, the techniques introduced here will be integrated into operational data systems at the National Space Science Data Center at NASA/Goddard Space Flight Center. These systems provide scientific data visualization and analysis tools for researchers in the Earth and space sciences. In order to improve the speed with which maps are rendered, the existing world coastline data bases and access software will be replaced with an SQT representation. Also, the software will be enhanced to support transparently an optional SQT representation of spherical data to facilitate user selection of regions for regridding, analysis and rendering. Later, some existing techniques for rendering data on a spherical surface will be reimplemented using an SQT-based approach.

The National Space Science Data Center's (NSSDC) next version of Common Data Format (CDF) will support non-cartesian spatial data structures, and will use SQTs heavily for managing indices for the locating archived data. We are also developing a Geographic Information System prototype (NSSDC Object-Oriented Geographic Information System, NOOGIS), a single system to manipulate multi-layered, heterogeneous data sets that are spatially indexed, such as sensor imagery and maps, easily and intelligently. This system is based upon a prototype system developed at the University of California at Santa Barbara, called KBGIS (Knowledge-Based Geographic Information System)[2]. Although much of the system internals are well developed, the system lacks an adequate user interface, and would benefit from being able to input and output imagery data in NASA formats. Enhancements are also planned for the user interface to handle domains with global data coverage. The SQT will be used organize the data to give a user rapid access to data layers located at a particular geographic regions on the Earth. This new access will give the user a browse capability to view high level details of global data layers on a regional scale.

Finally, plans are being made to enhance the Hubble Space Telescope Data Archive and Distribution propotype system [HSTDAD], resident at the NSSDC using SQTs as part of the user interface for spatial data selection. Careful integration of SQTs and various other technologies holds great promise for easy, effective and rapid data selection, browse and analysis.

6. REFERENCES

[1] C.M. Brown, "Fast display of well-tessellated surfaces," *Computers and Graphics* **4**, pp. 77-85, 1979.

[2] W.J. Campbell et al, "Intelligent Information Fusion for Spatial Data Management," to be published in the Proceedings of the 4th International Symposium on Spatial data Handling, July, 1990.

[3] J.D. Clinton, "Advanced structural geometry studies, part I: polyhedral subdivision concepts for structural applications," *NASA CR-1734/35*, 1971.

[4] H.S.M. Coxeter, *Regular Polytopes*, Dover, New York, 1973.

[5] G. Fekete, "Sphere quadtrees: a new data structure to support the visualization of spherically distributed data," To be published in the proceedings of the *SPIE/SPSE Symposium on Electronic Imaging Science and Technology*, February, 1990.

[6] A. Pugh, *Polyhedra: a Visual Approach*, University of California Press, Berkeley, California, 1976.

[7] A. Rosenfeld and J.L. Pfaltz, "Sequential operations in digital image processing," *Journal of the ACM* **13**, 4, 1966, pp. 471-494.

[8] H. Samet, "Neighbor finding techniques for images represented by quadtrees," *Computer Graphics and Image Processing* **18**, 1, 1982, pp. 37-57, 1982.

[9] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, Massachusetts, 1990.

[10] L.A. Treinish, "An interactive discipline-independent data visualization system," *Computers in Physics* **3**, 1989.

[11] L.A. Treinish, "The role of data management in discipline-independent data visualization," To be published in the proceedings of the *SPIE/SPSE Symposium on Electronic Imaging Science and Technology*, February, 1990.

[12] M.J. Wenninger, *Polyhedron Models*, Cambridge University Press, Cambridge, England, 1971.

[13] M.J. Wenninger, *Spherical Models*, Cambridge University Press, Cambridge, England, 1979.

APPENDIX

A.1 Propagating fragmentation

Subdividing a trixel may have an effect on the neighboring trixels. Suppose that, as in Figure 15, a trixel, *ABC*, should be subdivided, but its neighbor, *DBC* requires no

further subdivision. Then, with additional values at position A', DBC will have to be subdivided to satisfy the continuity criterion of surfaces. This requires that the surface along the common boundaries of trixels takes on the same values. Therefore DBC is subdivided, so that its new surface contour (now comprised of four linear patches) fits that of ABC. The parameters of the four new surfaces in DBC can be calculated without further sampling, since the values at A' and at the vertices of DBC are known. The two remaining values at the midpoints of DBC's boundaries not shared with ABC (C'' and B'') can be computed from DBC's parameters using linear interpolation. For our purposes the result of the interpolation is adequate, since DBC does not require subdivision. Thus, for each child of DBC the parameters are well defined, because values at all vertices are known. It is important to note that fragmentation of a trixel does not propagate beyond the trixel's neighbors, because the newly introduced points B'' and C'' are on the lines CD and DB, respectively, and therefore also in the same plane as the neighbors of DBC other than ABC.

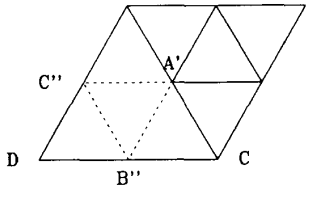

Figure 15. Effects of subdividing a trixel on its neighbor

A.2 Avoiding singularity

To compute the coefficients of the linear form, as in equation (1), ϑ and φ must be defined everywhere. However, there are two points at the poles, $\varphi = \pm\pi$, where ϑ is undefined. To avoid this problem of singularity, we introduce a right-handed orthonormal system of coordinates for each face of the basic icosahedron in such a way that φ and ϑ are both well defined within it. Each system's origin is at the center of the icosahedron. We define the orthonormal base $\{\mathbf{x}'_1, \mathbf{x}'_2, \mathbf{x}'_3\}$ as follows. $\mathbf{x}'_1$ is a unit vector from the origin to the center of the icosahedral face. $\mathbf{x}'_2$ is perpendicular to $\mathbf{x}'_1$ and parallel to a designated side of the icosahedral face (see Figure 16). $\mathbf{x}'_3$ is determined by the orthonormality constraint and the right-hand rule, that is, $\mathbf{x}'_3 = \mathbf{x}'_1 \times \mathbf{x}'_2$. Each of the twenty faces has a transformation matrix associated with it to transform global spherical to local coordinates. By global frame of reference we mean an object-centered, right-handed Cartesian reference frame, in which the object itself is defined. This transformation can be thought of as two successive rotations. The first one aligns the $\mathbf{x}'_1$ axis with the $\mathbf{x}_1$ axis by rotating around an axis perpendicular to both by the angle of their separation. The second one aligns the new rotated $\mathbf{x}'_2$ axis with $\mathbf{x}_2$ by rotation around $\mathbf{x}_1$. The combination of the two transformations changes global coordinates to the local (primed) frame. Each transformation can be carried out by multiplying the coordinates of a point in the global system by the product of the two rotation matrices. We can easily compute the rotation matrix $\mathbf{M}_\vartheta(\mathbf{a})$ which performs a rotation by ϑ around an arbitrary unit vector $\mathbf{a}$ by using the following relation

$$\mathbf{M}_\vartheta(\mathbf{a}) = \mathbf{a}\mathbf{a}' + \cos(\vartheta)(\mathbf{I} - \mathbf{a}\mathbf{a}') + \sin(\vartheta)\mathbf{I} \times \mathbf{a} \qquad (2)$$

where $\mathbf{I}$ is a 3 by 3 identity matrix, $\mathbf{a}$ is a column vector and $\mathbf{a}'$ is its transpose.

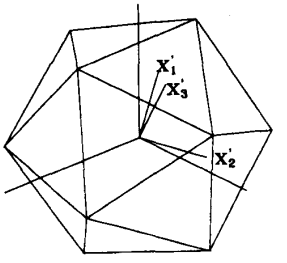

Figure 16. Local reference frame for a primitive trixel

A.3 Properties of trixel geometry

The of four fundemental properties of trixel geometry is discussed here.

Property 1: Two trixels with common parents are adjacent if and only if one of them is a "4" trixel.
This claim is easily verified by examination of the relative positions of the four offsprings "1" - "4" in Figure 7. From this it follows that every non-"4" trixel has a "4" neighbor. Because of the systematic naming of newly created vertices, we may talk about the relative orientation of adjacent trixels in terms of the vertex labels. The "4" offspring's relationship to the other three can be characterized by the labels of their common edge. For example, the "1" and "4" subtrixels share the $B'C'$ edge, but the "2" and "4" share $A'C'$. In general, there are a maximum of six ways two triangles sharing an edge can be oriented, if the same three labels are used within each triangle. In trixel geometry, however, we need to deal with only half as many.

Property 2: Two adjacent trixels can take on only three different relative orientations.
This can be easily verified by induction. Let us assume, that as in Figure 9, only three relative orientations exist between adjacent trixels. For each of the three cases, let us subdivide the two trixels, and examine the relative orientations of the adjacent trixels in the next generation. We observe, that all adjacencies are in one of the three configurations we postulated. To complete the argument, we note that Property 2 holds true for the children of the primitive trixel.

Property 3: The adjacent subtrixels of two adjacent subtrixels are of the same type.
This also follows from trixel construction, and is a consequence of Property 2. In Figure 9, for each of the three configuration of adjacent trixels the adjacent subtrixels are always of the same type. But because the subtrixels are themselves in one of the three configurations, the claim holds true for the offsprings of the offsprings, and so on.

Property 4: If two trixels are adjacent, then their names differ in exactly one symbol (but not conversely).
There are two cases to be considered. If the two adjacent trixels are siblings, then their names up to their parent are identical. Therefore only the last symbol in their name is different. This can only happen if one trixel is of type "**4**", as previously claimed. On the other hand, if the two trixels do not have a common parent, then they must have a pair of ancestors that share a parent. In this case, the ancestors' name falls under the above case, and by Property 3 all the adjacent offsprings will always have the same type, hence further symbols will be identical. As a counterexample to the converse of this claim, observe ⟨**144**⟩ and ⟨**444**⟩ in Figure 8.