A General Proof of the Multidimensional Binary Indexing Algorithm for Neighbourhood Calculations in Spatial Partition Trees

José Poveda, Michael Gould
Departamento de Lenguajes y Sistemas Informáticos
Universitat Jaume I
E-12071, Castellón, Spain
{albalade, gould}@uji.es
Phone: +34 964 728317
Fax. +34 964 728435

SUMMARY

We present a binary array encoding (location arrays) of the nodes in a spatial partition tree representing spaces of dimension k. This framework facilitates tree traversal for optimising access speed, and also supports simplified calculation of the neighbourhood of a subinterval of a particular partition. After defining the encoding we present a neighbour determination algorithm which extends work carried out by Samet (Samet 1984), (Samet, 1995) and others on quadtrees. The primary extension is that the encoding and the neighbour determination algorithm extend to arbitrary dimensions beyond the 2-d quadtree case. The application of this encoding to the creation of multiresolution terrain models for fly-over simulations was used in (Poveda, 2003) and in this paper a general demostration of the referenced algorithm is given.

KEYWORDS: *Spatial partition tree, binary location array, multidimensional indexing, multiresolution, neighbour calculation, terrain visualization.*

INTRODUCTION

The creation of large, moscaicked terrain models for fly-over simulation requires optimal data structures and indexing. Quadtrees represent the primordial hierarchical spatial data structure, whose common salient feature is that of recursive decomposition of 2-d space. Quadtree types can be differentiated by the following characteristics:

- The type of data used for its representation
- The principle guiding the process of partitioning
- The resolution (variable or not).

This data structure may be used for representing and handling point, curve, region and volume data. Decomposition of space can be regular at all the levels of the quadtree or non-regular, guided by the entry of the data to represent. The resolution of the decomposition (the number of times that subdivision is applied) can be set beforehand or can be guided for the properties of the input data. Depending on the specific application also a distinction can be made whether or not the structure is to define the border of a region, in the case of curves or surfaces, or if it is used to define its interior in the case of areas and volumes.

One of the principal variants of the quadtree data structure is the region quadtree, so much so that Samet (Samet, 1984) uses the terms synonymously. As an example we present the region in 1 that

initially we have represented as a binary array of 23 x 23 where the value 1 represents if a pixel is inside the region to represent and 0 if the pixel is found outside the region for a matrix representation of the region quadtree.

In the corresponding tree structure in figure 1, the root node corresponds to the complete array. Each son node represents a quadrant (labelled NW, NE, SW, SE) of the region represented by the node. The terminal nodes correspond to the blocks of the array in which it is no longer necessary to continue subdividing. A terminal node is represented in white or in black depending on whether its corresponding block is completely located inside the area to represent (all the elements of the array in this block contain the label "1") or that block is found completely outside the region of interest (all the elements of the array belonging to that block are labelled with "0").

These same spatial partitioning concepts extend to the third dimension in the representation of solids with octrees (Berg, 2000), although the encoding of blocks instead of quadrants becomes more complicated. In this paper we refer to spatial partition trees, in general, even though our initial examples treat the common 2-dimensional case using quadtrees. This is because the method and algorithm proposed here do not restrict spatial partitioning to 2-d but rather are extensible to spatial partitioning in arbitrary dimensions.

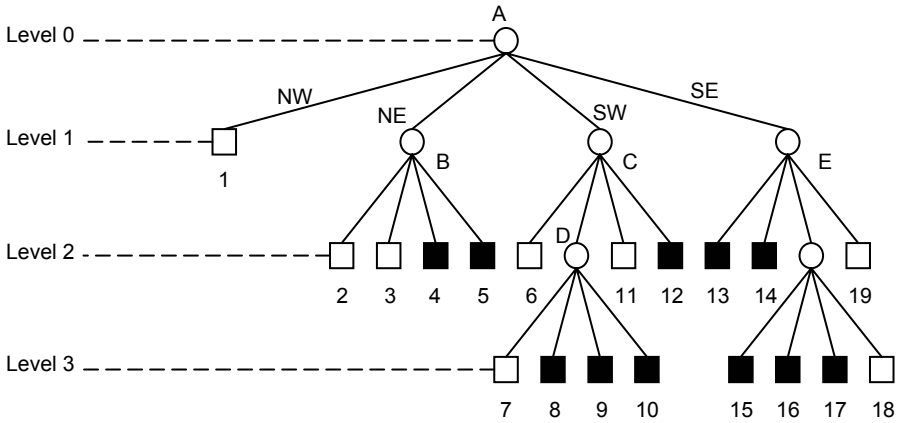

Figure 1: Example of quadtree representation. (Example taken from Samet, 1995).

Multiresolution terrain visualization

Let us look at an application case where neighborhood calculation using a quadtree structure is a crutial element in interactive terrain visualization. For this application we propose to optimize the visualizatin of an extensive Digital Terrain Model (DTM) utilizing management of levels of detail (LOD). With this goal we define a process which renders to the screen relevant information, that is, only the information which might be appreciated from the viewpoint of the user. The data structure considered to manage the MDT with multiple levels of detail is the quadtree. In the quadtree nodes we store the elevation values which constitute the MDT as a regular grid with different levels of resolution, where each level of the tree corresponds to a level of resolution of the terrain.

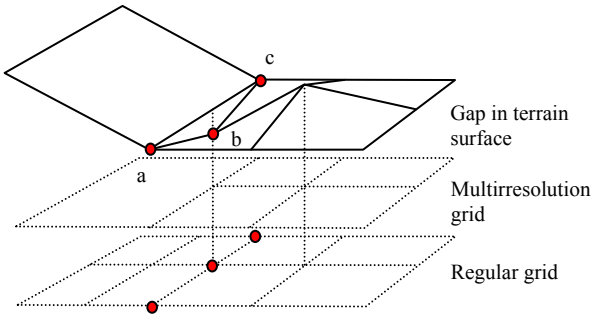

Figure 2: The union of differing levels of detail (LOD) here provokes a discontinuity in b: perspective view of the different layers utilized for generating the terrain.

In order to visualize the terrain, given a particular viewpoint, we traverse the quadtree and select the different levels of resolution represented by different levels in the tree. For the selection of nodes in the tree we use as main criterion the distance to the viewpoint, along with terrain roughness criteria. To insure against union of intervals with levels of resolution greater than 1, we run a balancing procedure over the pruned tree. Obtaining smaller regions after the pruning of larger adjacent regions, causes discontinuities (gaps) in the graphic representation of the DTM which cannot easily be corrected unless the difference in levels of resolution is no greater than one, see figure 2, in which case we make the correction interpolating the elevation of point b, taken from the region with higher resolution, which provoked the discontinuity, to the segment *ac* of the neighboring region with lower resolution.

For this process, therefore, it is necessary to calculate the intervals which are neighboring a given interval. Analogous to the multiresolution management, mentioned for the simplification of terrain geometry, it is also necessary to manage the simplification of the texture which overlays that geometry.

BINARY ENCODING PARTITIONS

Frequently GIS-related operations will require rapid access to a specific node in the tree. This normally requires traversal from root node downward through the tree, passing from node to node depending on the area representing the quadrant in question; this in turn supposes that position information is carried for each node. Normally quadrants are indexed according to a hierarchical base-4 ordering such as that proposed by Morton (the Z order). While this ordering simplifies operations in 2-d space and also facilitates human description in terms of four cardinal directions, a simpler referencing method is desirable as dimensionality increases. Here we describe such a method.

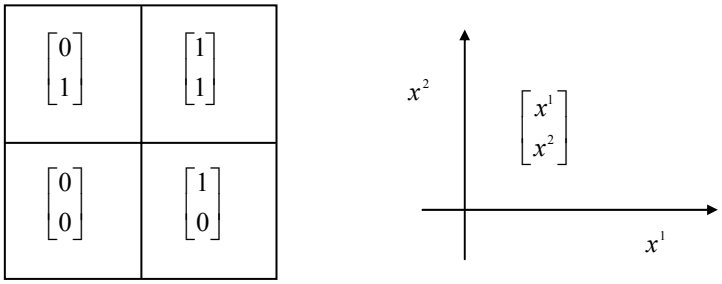

Figure 3: Binary array encoding in the 2-d case.

To increase access simplicity it is possible to apply binary encoding [4] to nodes so that these codes determine the traversal path down the tree. Rather than suppose four cardinal directions we encode each dimension with a 1-bit coordinate, describing the direction as positive or negative with respect to a given reference system.

For example, in two dimensions the positive directions --up and to the right (or East)-- are represented by a 1, while the negative directions --down and left-- are represented by 0. Consider the example in figure 3, where the "northwest"19 quadrant at level 1 is assigned a 0 (negative direction) in the x1 dimension and a 1 (positive direction) in the x2 dimension. In this 2-d case the assumed reference system has its origin in the central point of the area where the quadrants meet. The example in figure 4 expands on this binary encoding, showing the notation to three levels.

BINARY LOCATION ARRAY

Taking into consideration the binary encoding of each dimension we define, as shown in figure 3, an array representation of each subset region which communicates both the location in the tree structure and, implicitly, the path to reach the node associated with that region.

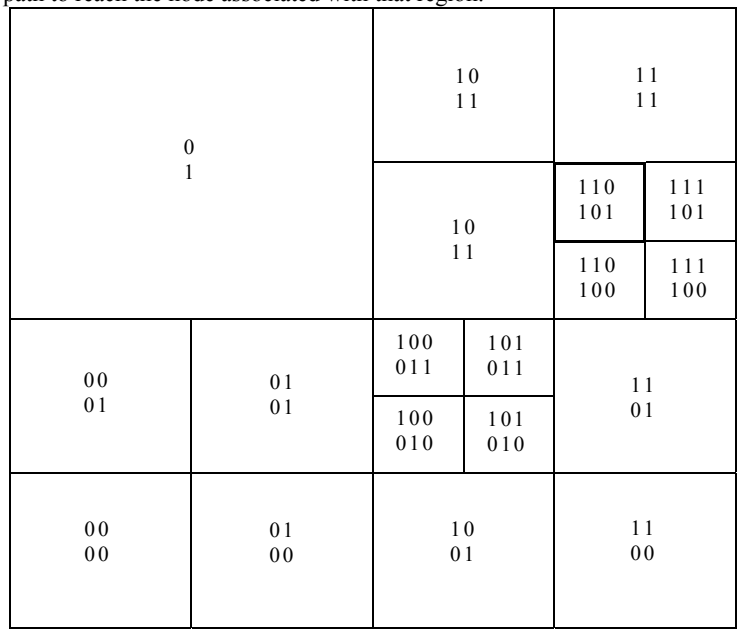

Figure 4: Binary location arrays of a quadtree to the 3rd level.

The location array associated with a given node (here quadrant) is defined as the array of the father node, then adding on the right a new column vector with its own encoding according to the system in figure 3. The 3rd level node highlighted in figure 4 is thus represented by the 1,1 of the grandfather, the 1,0 of the father and then adding the 1,1 for its own level. Figure 5 shows additional examples to help clarify the encoding proposed.

[19] While it is all too tempting to use cardinal references, we again underscore that the positive/negative direction notation applies beyond the 2-d case equally to all dimensions.

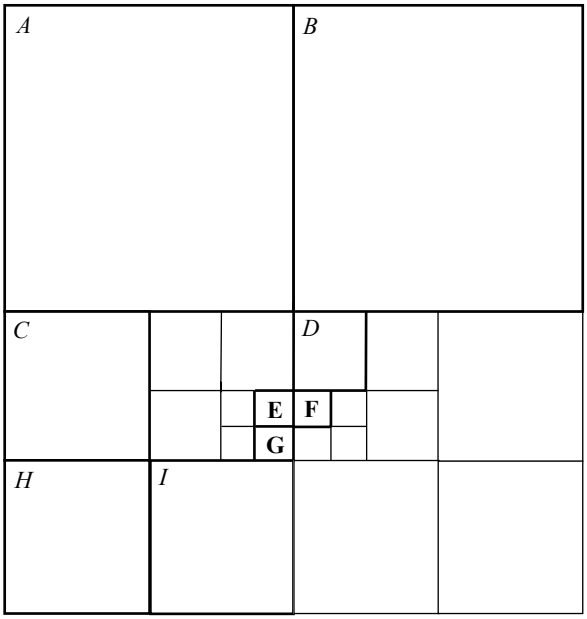

Figure 5: Region quadtree example.

The location array representation of the highlighted regions in figure 5 (four levels in 2-d space) would be the following:

$$A = \begin{bmatrix} 0 \\ 1 \end{bmatrix};\ B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix};\ H = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix};\ I = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$E = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix};\ F = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix};\ G = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Encoding higher dimensions follows the same principle described above. For example the following array P shows 5 levels of nodes (columns) in 4-d space (rows).

$$P = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Because the location of the quadrant[20] (node) is defined by this simple binary location array, in section 4 we describe a sort of bit shifting on these arrays for simple neighbour calculation in any arbitrary dimension.

MULTIDIMENSIONAL NEIGHBOUR CALCULATION

Neighbor calculation is a critical operation for locating and reading into memory the next terrain file (tile) to be displayed during a real-time terrain fly-over simulation. To achieve maximum data access efficiency and simplicity it is useful to be able to reason about neighbour locations in the tree, identifying all possible neighbours of each node. These neighbourhood references are useful for tasks such as database paging or for adjusting resolution in a geometric model during an interactive flight over terrain (De Berg, 2000), (Falby et al., 1993), (De Florian & Puppo,1995) (keeping in mind that the fly-over "pilot" might change direction 1 to 359 degrees). If the terrain model includes multiple levels of resolution as a function of attributes such as distance to the viewer, then for each frame (and fly-overs typically aim for 10 frames per second) it would be necessary to recalculate the spatial partition tree and the neighbour relations of each node because quadrants of distinct levels of decomposition can cause discontinuities (gaps) which must be corrected by adjustment to the lower resolution (see examples in figure 5). Because these applications will need to continuously recalculate these neighbour relations, it is essential that the algorithm for doing so deal efficiently with neighbours at multiple resolutions.

As a solution to clarify and optimise neighbour identification, we utilize the location array described above, in a simple algorithm of linear computational cost with respect to the depth of the tree.

To calculate the neighbour of *E* (in figure 5) in the positive direction at dimension 1 and at the same level of resolution (that is to say, quadrant of equal size or node at the same level), we begin with the location array of E and we visit each element, indicated by a dot, in the top row from right to left. Alternating the bits as we go along the first 1 becomes a 0, the second 1 a 0, the third also a 0, then the 0 a 1. When we negate a 0 and it switches to 1, at any point in the row, the process stops and the resulting location array describes the neighbour found, in this case *F*.

$$E=\begin{bmatrix}0 & 1 & 1 & \dot{1}\\ 0 & 1 & 0 & 1\end{bmatrix}\Rightarrow\begin{bmatrix}0 & 1 & \dot{1} & 0\\ 0 & 1 & 0 & 1\end{bmatrix}\Rightarrow\begin{bmatrix}0 & \dot{1} & 0 & 0\\ 0 & 1 & 0 & 1\end{bmatrix}\Rightarrow\begin{bmatrix}\dot{0} & 0 & 0 & 0\\ 0 & 1 & 0 & 1\end{bmatrix}\Rightarrow\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 1\end{bmatrix}=F$$

Now, if we continue applying the same algorithm we may calculate at the same level of resolution the neighbour of *E* in the *negative* direction in the second dimension ("down"), producing the location array which represents region G (see figure 5). This time we have alternated the bits in the second row (representing the second dimension), until a 0 is obtained, at the first move in this case.

$$E=\begin{bmatrix}0 & 1 & 1 & 1\\ 0 & 1 & 0 & \dot{1}\end{bmatrix}\Rightarrow\begin{bmatrix}0 & 1 & 1 & 1\\ 0 & 1 & 0 & 0\end{bmatrix}=G$$

In general terms, then, we can state that in order to search for a neighbour in any *positive* direction, the halt condition in the bit shifting is the negation of any 0 to a 1. Equally then, to search for a neighbour in any *negative* direction the halt condition is the negation of any 1 to a 0.

The principal advantages of this neighbour calculation algorithm are its generality, with respect to spatial dimensions, and its simplicity. A representation of the algorithm (for either direction) would be the following.

[20] Use of the term quadrant refers to any interval of the spatial partition, at any dimension, and associated in the tree as a node.

Algorithm CalculateNeighbour (A[i][j], ± e_k)

Input. Location array A[i][j] of the region quadtree of interest, in direction e_k, where r is the level of resolution.
Output. The array of the neighbour encountered in direction e_k

```
1.- if (ek >0) then
2.-  while (A[k][r]==1)||(r==0)
3.-        A[k][r] <-not(A[k][r]);
4.-        r<-r-1;
5.-  A[k][r] <-not(A[k][r]);
6.- else
7.-  while (A[k][r]==0)||(r==0)
8.-        A[k][r] <-not(A[k][r];
9.-        r<-r-1;
10.- A[k][r] <-not(A[k][r]);
11.-return A
```

DEFINITION

Given a quadtree partition in *k*-dimensional space, an interval I of that partition at level of resolution n + 1, and its associated location matrix as defined as follows $M[I]=\begin{bmatrix} x_0^1 & x_1^1 & \dots & x_n^1 \\ x_0^2 & x_1^2 & \dots & x_n^2 \\ & & & \\ x_0^k & x_1^k & \dots & x_n^k \end{bmatrix}$

we define the direct sum of *M[I]* with the vector $\overline{w}=\begin{bmatrix} w^1 & w^2 & \dots & w^k \end{bmatrix}$ of dimension *k* as follows

$$M[I]\oplus\overline{w}=\begin{bmatrix} x_0^1 & x_1^1 & \dots & x_n^1 \\ x_0^2 & x_1^2 & \dots & x_n^2 \\ & & & \\ x_0^k & x_1^k & \dots & x_n^k \end{bmatrix}\oplus\begin{bmatrix} w^1 \\ w^2 \\ \\ w^k \end{bmatrix}=\begin{bmatrix} x_0^1 & x_1^1 & \dots & x_n^1 & w^1 \\ x_0^2 & x_1^2 & \dots & x_n^2 & w^2 \\ & & & & \\ x_0^k & x_1^k & \dots & x_n^k & w^k \end{bmatrix}$$

Note that the location matrices of the son nodes of any given node may be expressed as the direct sum of of the location matrix of the father node plus the binary coding of each of the sons, summed as column vectors.

NEIGHBORHOOD THEOREM

The neighborhood theorem demonstrates the Poveda-Gould neighborhood algorithm for quadtree regions. The neighborhood algorithm is developed as follows:

Algorithm (Calculate neighbor in direction $+x^i/-x^i$)

1.- First element processed: the last element in the row i, M(i,j) in the location matrix.

2.- **IF** the processed element is a 0/1, negate the element and the process terminates. The resulting matrix is the location matrix associated with the neighbor node in the direction $+x^i/-x^i$.

3.- **ELSE** the processed element is negated and repeat step 2 with the next element to the left in the row.

DEMONSTRATION

Given the interval I in *k*-dimensional space, which is determined by the location matrix *M[I]*, we calculate the neighbor *I′* in the positive direction and first dimension, which does not imply loss of generality. The location matrix of I may be written as $M[I] = M[P(I)] \oplus \begin{bmatrix} x^1 \\ x^2 \\ \\ x^k \end{bmatrix}$

where P(I) identifies the father interval of I. Therefore, if $x^I=0$, *I′* will have the same father as I and in addition by the encoding produced it is the case

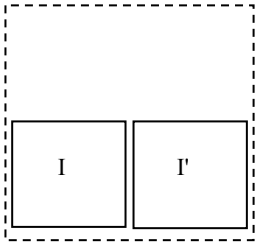

Figure 6: Neighboring intervals with common father

the location matrix of *I′* is equal to $M[I'] = M[P(I)] \oplus \begin{bmatrix} \overline{x^1} \\ x^2 \\ \\ x^k \end{bmatrix}$

if $x^I=1$, *then* would not have the same father as I, however due to its encoding it would be true that

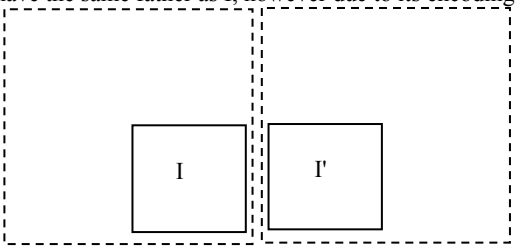

Figure 7: Neighboring intervals with neighboring fathers

therefore, the location matrix of I' is equal to $M[I'] = M[P(I')] \oplus \begin{bmatrix} \overline{x^1} \\ x^2 \\ \\ x^k \end{bmatrix}$

and then what remains to be known is the value of $M[P(I')]$ which is what we considered at the beginning however with a lower resolution, as seen in the next figure.

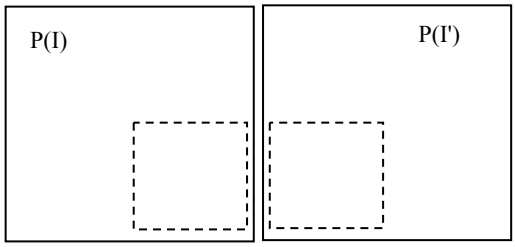

Figure 8: Neighbor intervals P(I) and P(I')

We procede iteratively until reaching $x^1=0$ at which point the location matrix has been calculated explicitly.

REFERENCES

Samet, Hanan, "The Quadtree and Related Hierarchical Data Structures", ACM Computing Surveys 16, 1984, pp. 187-200.

Samet, H. “Applications of Spatial Data Structures”, Addison-Wesley, 1995.

M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, “Computational Geometry, Algorithms and Applications”, Springer, Second Edition, 2000, pp. 291-303.

Goodchild, M.F. (Ed.), 1990. “Quadtree algorithms and spatial indexes”. Technical issues in GIS. NCGIA Core Curriculum. Unit 37.

De Berg, M., Dobrindt, K. T. G., "On levels of Detail in Terrains", 11th ACM Symposium on Computational Geometry", 1995, pp. 1-15.

Falby, J. S., Zyda, M. J., Pratt, D. R., and Mackey, R. L., "Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation", Computer and Graphics 17, 1993, pp. 144-151.

De Floriani, L. and Puppo E., "Hierarchical Triangulation for Multiresolution Surface Description", ACM Transactions on Graphics 14, 1995, pp. 363-411.

Poveda, J., Gould M., Sevilla, J. “GeoIndex Project: A virtual window to Geographic Informations Systems”. In Rault, J.-C. (Ed.): La Lettre de l'Intelligence Artificielle, Proceedings of the International Conference on Complex Systems, Intelligent Systems & Interfaces (NIMES'98), Nimes, France. Nimes 1998.

Schee, L. H., Jense, G. L., "Interacting with Geographical Information in a Virtual Environment", Proc. Joint European Conference on Geographical Information. The Hague, NL, 1995, pp. 151-156.

Schroder, F. and Rossbach, P., "Managing the Complexity of Digital Terrain Models", Computer and Graphics 18, 1994, pp. 775-783.

José Poveda, Michael Gould, "Multidimensional Binary Indexing Neighbourhood Calculations in Spatial Partitions Trees". Proceedings GEOPRO 2003.