

Design without Precedents

H. Koppelaar¹, R.A. Vingerhoeds^{1,2}, D. Chitchian¹

¹ *Delft University of Technology, Faculty of Technical Mathematics and Informatics*

² *University of Wales Swansea, Department of Electrical and Electronic Engineering
Singleton Park, Swansea SA2 8PP, United Kingdom.*

I. EVOLUTION OF DESIGN AUTOMATION METHODS

New methods of automating design generation differ greatly from the ways AI researchers have attempted to elicitate and implement design knowledge. The latter tradition is coined by the so called 'knowledge based approaches'. A good example for floor plan design can be found in (Zandi 1992).

Using iterative algorithms (e.g. artificial neural networks), this programming work can be relieved, carrying out learning of patterns. During the learning process the parameters of the iterative algorithm are gradually adjusted until a particular input produces the desired output. Advantages of this approach can be found in the non-necessity to specify the knowledge to be implemented, or how a problem is solved. On the other hand, the knowledge captured in a neural network is not explicit and therefore not explicable. For floor plan design this approach was taken by (Zandi and Koppelaar 1992).

Disadvantage of both approaches is that they both are specific for the problem at hand and it is difficult to adapt them to a wide range of varying problems. Each set of expert rules is semantically bound. Each set of learned connections in a neural net is implicitly specified by its learning set.

To pursue automation of design tasks in a wide range of domains, an approach of evolving software is proposed here. The software starts from a local starting point and evolves, without any precedent, to a design. In the initial application explained in this paper, design of a floor plan is studied. The software breeds from an atomic locational egg, with a seed operating upon the egg. The 'fittest' floor plan proposals in the breed—in our case, the ones which come closest to the pre-set privacy and community design goal—are allowed to 'survive', 'breed' and 'spawn' the next generation. From a starting population of one single location it takes over a number of generations to automatically design this floor plan:

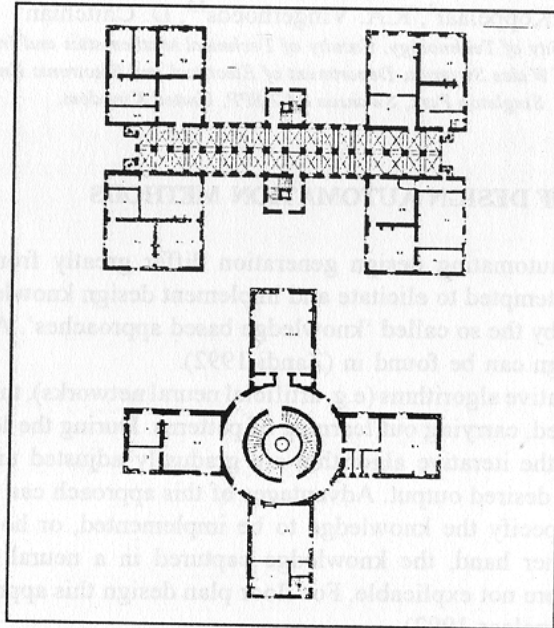


Figure 1: Samples of global designs

2. HIERARCHICAL STRUCTURES

To reduce computational effort of design, it is necessary to know in advance which atomic structures enable our design program to construct a large building with a target performance.

In order to reduce computational complexity the common approach of hierarchical description of floor plans is introduced. A common problem with very large floor plans is to find its backloth. Ad hoc attempts would give a description too fine with overwhelming detail, or too coarse. Hierarchical schemes attempt to provide exclusive classes to partition data-space. As this is often not needed, a backloth does not need be covering. Omniscience over data-space is not needed. A hierarchical scheme should structure data-space where necessary. Hierarchical schemes employ intermediate levels. The building blocks of the intermediate levels should be such that they are useful entities between a higher and a lower hierarchical level.

A hierarchy is a structure that clusters a number of elements in one level into a single aggregate on the next higher level. Aggregates themselves may be clustered again into higher level aggregates and so forth. Each element or aggregate

belongs only to one aggregate at the next higher level, thus the graph of a hierarchy is a (not necessarily binary) tree. Aggregates summarise properties of their constituting elements. By varying the aggregate level, we may look at data at various levels of detail (resolution). A distinction has to be made here between spatial and semantic hierarchies. Semantic hierarchies are often used in artificial intelligence, natural language understanding and object oriented systems. The different levels in a semantic hierarchy correspond to various abstraction levels. Semantic hierarchies are often used to inherit properties from higher level aggregates to descendants. For example, a dog is known to be an animal, as all birds are, etc.

In this paper the focus is on spatial hierarchies, cataloging and clustering catalogue space. Object-based hierarchies and space based hierarchies can be distinguished. Space-based hierarchies can exploit the regular subdivision characteristics by using simple algorithms, as they do not presuppose known, precisely located object boundaries. On the contrary, segmentation is exactly the kind of thing one wants to extract from a raw design. Also, designs are usually supplied in a sort of raster, which constitutes the bottom level of the spatial hierarchy. All these raster elements will usually have the same size and shape, so a raster-based spatial hierarchy is most straightforward.

Applications and representations for hierarchical data structures are discussed by using the following topics:

- *Tiling elements*: The basic building blocks for building data bases.
- *Aggregation strategies*.
- *Locational Codes*.

3. TILING ELEMENTS FOR NEIGHBOURHOODS

The word 'representation' has been used in AI to denote computational structures of different types. If C denotes an arbitrary set of objects and c denotes a representation of C , then c may represent C merely by virtue of the fact that we endow it with a certain interpretation. Alternatively, c may represent C in a more formalised way. We say that c is 'loosely coupled' only if it has a computational relationship with C . It is 'classificatory' if it forms a predicate which correctly gives descriptions according to whether or not they are in C , it is 'generative' (or 'deeply coupled') if it is a program which generates C and it is 'deterministic' if it generates/classifies (elements of) C perfectly, or 'approximate' if it does so to some degree of accuracy.

A designer trying to invent a suitable representation strives for simplicity. Systems should simply be modelled to interact locally, where locality is expressed by the neighbourhood connections of the design graph. With this graph, the overall model is built incrementally. From these local operators emerges the complexity of the overall model.

Definition

A neighbourhood representation is a coupled representation of C with a certain type c in a data language D.

Any loosely coupled representation forms a neighbourhood representation if there is an interpretation c in which the represented set C is defined in terms of the points inside non-degenerate regions of an n-dimensional space. In order to create hierarchies, data-space has to be subdivided (and then aggregated). The subdivision will be such that the elements will completely cover n-dimensional space, without gap or overlap. There are a large number of choices for such space-filling 'raster' or 'tiling' elements (see Grunbaum and Shephard 1987 and Escher 1989), with direct consequences for processing efficiency and effectiveness.

Once a tiling for data/floor plans has been selected, an aggregation strategy must be devised in order to invoke structure to collections of elements. There is a whole list (e.g. see Grunbaum and Shephard 1987) of properties that various structures and hierarchies have and in order to make a well-argued choice, these have to be measured and compared in the knowledge acquisition phase.

In processing entities and their aggregates, the hierarchy to access them can be used. But it would be much faster to access them directly. Such a spatial address can be used to process data very efficiently (Chisvin and Duckworth 1989). Similarly, hierarchical locational codes immediately show what region of space a certain cell or aggregate is in.

In a tree-structure, each box is now given an address (its locational code) and each box may contain a number of properties. The contents of boxes are used in semantic hierarchies. Semantic indexes such as "privacy" and "community" of a building, equivalent to a locational code, are developed in this paper. In the case of a tree-storage scheme, entities can only be accessed by traversing the hierarchy, either top-down or bottom-up, i.e. by semantic proximity. Locational codes, however, give direct access to information: it reduces search. Experimentation with the spatial metaphor (Jones and Dumas 1986) shows the applicability for user interfacing.

Once locational codes for a hierarchy have been obtained, they can be used for other applications than access of entities. Rotations, projections and other operations can be defined that act on locational codes, without the need to refer to a computationally complex relational data structure such as the hierarchy itself.

Examples of locational codes:

- street side known by house numbers
- a hierarchical locational code for precise location
- hierarchical locational code for a quadtree

Algorithms using locational code arithmetic typically require fewer operations than their Cartesian co-ordinate equivalents, although each operation is more complex in its meaning. The integer arithmetic involved in calculating distances is highly parallelisable and hence special purpose “locational arithmetic computers” like hypercubes are very fast tools for data/floor plan processing. This is the area of applied discrete geometry.

One of the earliest application areas of hierarchical data structures is the encoding of two-dimensional images, using so-called quadtrees. A quadtree consists of a ‘root’ square, which is subdivided recursively until each square satisfies a primitive condition.

Quadtree encoding is particularly useful for images in which the objects are clustered, i.e. data is not spread more or less homogeneous in the plane. Because only the ‘black’ bits in a drawing need to be stored explicitly, this can result in significant reductions of memory requirements. Especially for data/floor plans in higher dimensions, a hierarchical encoding scheme can offer significant savings, since data is usually going to be sparse and storage without some compression scheme would be immense.

4. TESSELLATIONS AND HIERARCHIES

The four aspects of a theoretical framework are now presented:

- Space tessellations are presented and evaluated.
- Aggregation strategies for various tessellations. A number of aggregation properties are defined which influence practical algorithm design.
- The idea of locational codes for aggregations, concentrating in particular on rectangular and hexagonal cases, since these are the simplest and most effective, respectively.
- How can “normal” and useful arithmetic operations be performed on locational codes if they are treated as vectors.

4.1 Space tessellations

The task to be done is to process n-dimensional spatial data with as result a tiling of space without gaps or overlaps. Various authors call lattices arrays, grids, mosaics, tessellations and tilings and the elements pixels, tiles or primitives. Rectangular lattices are not the only choice and they are not always the best. Tiling is a tool to hold spatial data and not an end in itself. There are the computer science constraints of time and space efficiency. Tiling should be as effective as possible when a real, continuous data/floor plan is being sampled. The following paragraphs will present a number of alternatives for space filling tilings, and criteria to compare them. For practical purposes, the elements of the tiling should all be the same, i.e. they should be 'isotropic'. This simplifies data structures and algorithms. Furthermore, favourable quantisation characteristics are required and the sampling should be as rotation invariant as possible. A class of tiling elements that has been widely studied are the regular polygons: these have equal sides and interior angles. The simplest regular polygon is an equilateral triangle, followed by the square, the regular pentagon and so forth. Many sided regular polygons approximate circles. If regular polygons are to be used, the plane can only be covered using equilateral triangles, squares or regular hexagons. Of these, the rectangular and hexagonal tessellations have been studied and used extensively. Strangely the triangular tessellation has received far less attention. If the single-tile constraint is relaxed, tessellations with two or more regular elements can be found. It can be shown that there cannot be less than two or more than six regular polygons around a vertex.

A dual of a tessellation is another tessellation, with vertices at the centroids of the original tiles and edges crossing the original tile edges. The tiles of the Laves nets are usually not regular polygons. The nomenclature for the Laves nets is based on the number of edges coming to a vertex taken in order around the basic polygon.

By far the most popular tiling in two dimensions is the rectangular tiling, which is generally preferred because:

- It is simple to implement and it obviously can be generalised to higher dimensions without difficulty.
- In contrast to the hexagonal tiling, it is locally sub-dividable with the same tile shape as the original tile. This property is used for adaptively subdividing data structures such as the quadtree or the octree.

However:

- Rectangular tilings suffer from a topological problem: the figure-background paradox. In a rectangular tiling each cell has four axial nearest neighbours and four diagonal neighbours at a greater distance. For image analysis, the genus—the number of holes—of an object is often required. But in a binary image of an object with a hole in it, the background outside the object and inside the hole may still be connected, leading to a false determination of the genus of the object and thus wrong object classifica-

tion (Koenderink 1990). There are no such problems in a hexagonal lattice however, since each cell has six nearest neighbours at equal distance.

- Another problem with rectangular tilings is their bad sampling behaviour. They produce more sampling errors than for example the hexagonal tiling. To get comparable performance, the rectangular tiling requires higher resolution, i.e. more cells.

The hexagonal lattice has been preferred for image processing because:

- It has favourable sampling characteristics.
- The fact that all six neighbours of a hexagonal cell are at a uniform distance. This is important for so-called thinning or skeletonising algorithms employed in pattern recognition systems, for example in optical character recognition. Such thinning algorithms 'eat' away the boundaries of objects until the objects themselves are only one pixel wide. Such skeletonising operations do not affect the number of holes or joints in an object, so they are also called topology-preserving thinning operations. It is possible to put a simple processor element behind each pixel that is wired to carry out a local operation involving only its own pixel value and those of its neighbours, so that one thinning iteration can be carried out across the whole picture in parallel.
- The same uniform distance characteristic of hexagonal tilings also makes them better candidates for geometrical calculations like the first and second derivative for edge detection algorithms or relaxation methods, if a hexagonal grid is used for relaxation.
- The hexagonal lattice is more isotropic than the rectangular lattice, i.e. less sensitive to rotations, because of its high degree of symmetry. This means that the hexagonal lattice produces on average less quantisation noise (i.e. for circular figures) than the rectangular lattice.
- The hexagonal tile is the most economical choice for tessellating the plane, since it has the highest area to circumference ratio. This criterion also applies to the truncated octahedron in three-dimensional space and to the n-permutahedron (Lucas 1979) in n-dimensional space.

However:

- Hexagonal cells cannot be recursively subdivided into smaller cells of the same shape. The consequence for building hierarchical data structures using hexagonal tessellations is that these have an implicit bottom, a limit on precision which requires extra attention.

- For hexagonal arrays and their n-dimensional analogies the address arithmetic is more complex than for rectangular arrays, at least on binary computers. However, the generalised balanced ternary number system can be used (Knuth 1969).

Nature has also made use of the hexagonal rasters as well: insect's facet eyes which consist of many cones with hexagonal bases. Honey bees with their preference for hexagonal cells in the honeycomb have in fact found the best and most efficient form there is. Bees could have built their cells with round walls, but with round or nearly round (pentagonal, octagonal) cells there would always be some unused gaps left over. With triangular, rectangular and hexagonal cells the plane is covered without gaps. Of these three shapes, the hexagon has the smallest circumference for equal areas. Therefore bees require the least building material for a hexagonal honeycomb. Furthermore, the round larve that develops in the cell fits better into hexagonal than in rectangular or triangular cells.

The fact that triangular lattices have received so little attention in the literature is surprising and disappointing: they have properties from both the rectangular and hexagonal grids, and these might be exploited to get 'the best of both worlds':

- It has been shown that the triangular grid is a special case of a (skewed) rectangular grid (Peterson et al, 1962; Haas, 1985).
- Like rectangular lattices, triangular grids can be subdivided locally. This useful property allows higher resolution sampling in parts of an image without globally re-sampling at a higher frequency. Hexagonal grids cannot be subdivided like this.
- Triangular grids and hierarchies share a number of favourable hexagonal grid characteristics by virtue of 'embedding': six adjacent triangular tiles around a central vertex form a hexagonal tile. Clusters of such 6-groups can be used to form hexagonal grids and hierarchies.
- It is noted that triangular grids can be constructed by intersecting infinite straight lines, just like rectangular grids. This is not the case for hexagonal grids.
- There is the slight problem of adjacent triangular tiles having opposing orientation. However, in the addressing schemes introduced below, this property can easily be incorporated using an even/uneven horizontal address case distinction.

4.2 Aggregation Strategies

Apart from aiming for data compression another important reason for constructing hierarchies of space tessellations is to allow algorithms to process data at the 'right' level of detail: the hierarchy acts like a zoom-lens as there are only few elements in the upper levels of the hierarchy, so processing is fast there and decisions can be made quickly on where to continue or elaborate. If necessary, algorithms can zoom in on data in some locations, for example to increase precision in tracking features. Just as for choosing the space tessellation itself, a number of choices are available for hierarchies, even for one particular type of tessellation:

- *Isohedrality*. A tessellation is termed isohedral if all the tiles are equivalent under the symmetry group of the tiling, i.e. the number of edges emanating from each vertex remains invariant under the appropriate mirroring axes of the grid.
- *Similarity*. The tiles at levels are 'similar' if they have identical shape, i.e. the tiles at one level are scaled images of those at another level. The similarity need not be geometrically exact, as long as the topology (number of neighbours and adjacency relationships) remains the same.
- *Progression*. This criterion is only defined for similar hierarchies. It is the amount of rotation between a tile at level k and a tile at level $k+1$.
- *Regularity*. A tiling is regular if its tiles are regular polygons.
- *Orientation*. A tiling is said to have uniform orientation if all the tiles have identical orientation. Two tiles have the same orientation if they can be mapped onto each other by translations of the plane, without rotations or reflections. Generally, the orientation number of a tiling is the number of distinct tile orientations.
- *Convexity*. The convexity of a tile is the difference in area between the tile and its closed convex hull, expressed as a fraction of the area in the tile.
- *Adjacency*. Two tiles are considered to be neighbours if they are adjacent either along an edge or at a vertex. A tiling is uniformly adjacent if the distances between the centre of one tile and the centre of all its neighbours are the same.
- *Isotropy*. The isotropy number of a tiling is the largest order of rotational symmetry of the tiling (where a rotation of order n is a rotation by $360/n$). The isotropy number is a measure of how dependant the data structure is to rotations in the input data/floor plan. Ideally, the data structure built up from an input data/floor plan should be entirely independent of the orientation of the sampling grid with respect to the input. This is evidently not the case and every grid and hence hierarchy has 'preferred' directions along which sampling works best.

- *Aperture.* This is the number of tiles needed to assemble one aggregate. The higher the aperture, the easier it is to capture different 'pattern types' of an aggregate, which may be generated at any level of the hierarchy (Gibson and Lucas 1982). On the other hand, high apertures mean there is a lot of difference in covered space between levels of the hierarchy, which in turn means that the 'zoom-lens' property of the hierarchy is very discrete: only a few useful levels of the hierarchy exist.
- *Sphericity.* The sphericity of a tile is the difference in area between the smallest circumscribed and the largest inscribed circle which can be drawn, expressed as a fraction of the area of the tile.
- *Democracy.* Democracy among level t tiles exists when in the level $k+1$ tile it is impossible to differentiate between the level k tiles. This is a property which may or may not be an advantage, depending on the application. Thus the polygonal coherence afforded to 'pattern types' obtained by having a clear central tile is an advantage of undemocratic molecular tiles. This is also true for calculating properties such as gradients. However, it is an advantage to be able to treat all tiles with the same algorithms democratically. Finally, undemocratic tilings are usually preferable because they avoid the complicated addressing structures which democratic tilings often need to maintain relational support.

Previous criteria by Bell are useful reference to evaluate existing and new alternatives. A second effort towards a taxonomic treatment of hierarchical spatial structures is the nomenclature: short and precise notations should be developed that specify an n -dimensional tessellation and its aggregation strategy.

Next step would be the specification of the aggregation strategy (assuming that it is the same at all levels of the hierarchy, which it always is for practical reasons). This could be done by enumerating all tiles on level i that constitute a single level $i+1$ tile. They could be listed using their co-ordinates in a suitable integer co-ordinate system of the tessellation. A co-ordinate system that spans n -space is always possible although it need not equal the standard orthogonal co-ordinate system in n -space. This method is not yet satisfactory: the syntax is cumbersome, not all details have yet been specified and the aggregation is not yet specified in a rotation invariant way. Maybe a method based on tile adjacencies could be used instead of absolute co-ordinates. This would at least solve the last problem.

4.3 Locational code schemes

Locational codes number space by assigning each tile of an n-dimensional tessellation a unique address. The idea was first published by Gargantini (1982) for quadtrees. She called the new way of handling such a spatial data structure a linear quadtree, for the locational codes were stored in a linear list. Extensions to higher dimensions are straightforward. A similar system for hexagonal tilings was described by Gibson and Lucas (1982). It is based on the generalised balanced ternary (GBT) number system (Knuth 1969). The addressing scheme can be extended to n-dimensional tessellations of space using permutahedron packings (Lucas 1979).

The properties of locational codes are: Each tile of a tessellation in n-space is assigned a single, unique address. In this way, n-dimensional space is effectively transformed into one-dimensional space (the space of non-negative integers). Each aggregate in a spatial hierarchy also gets a single, unique locational code. The numbering of the atomic and molecular tiles is systematic and consistent to allow fast algorithms and useful arithmetic operations. The locational codes of all the tiles that constitute an object can be stored in a linear list. Tiles that are not part of the object(s) are not included in this list. Such linear lists are typically much smaller than an alternative pointer based structure, so locational codes are an effective way to compress data. By transforming discrete space into numbers, it becomes possible to apply meaningful arithmetic operations such as addition, complement and multiplication to those numbers. Space itself becomes an algebraic structure. This allows rotations, translations, and projections of spatial hierarchies. Such operations are not impossible, but more complicated with pointer based structures. Also, arithmetic operators can be implemented in dedicated hardware which can process many locational codes in parallel.

With little loss of generality, the following discussion of locational code schemes is based mainly on quadtrees. Exactly analogous schemes exist for the hexagonal hierarchies. In a locational codes for a rectangular 9-tree a centre cell is given digit value 0, surrounding cells are numbered in clockwise order with digit values 1 to 8. Note how the most significant digit of each code specifies the coarse location, and the least significant digit value specifies precise location. To explain the difference in style between the dynamically allocated pointer based representation of a quadtree and the single-list linear quadtree, and its rectangular subdivision, a binary image (for simplicity) is sampled with a rectangular grid. Cells that contain more black than white are considered black, the other ones are considered white. Then black regions are merged, creating squares of maximum size.

The construction of the linear data structure proceeds as follows: the whole image is scanned and the (x,y) co-ordinates of all black pixels are turned into a locational code. Simple algorithms for converting n-dimensional Euclidean co-ordinates into locational codes exist, examples will be presented below. Note that while locational codes obviously have a lexicographic order (e.g. 1201<1202<1300), converting pixels in scan-line order and appending their locational codes into the list will usually not yield a sorted list. So as a second step, the list of locational codes has to be sorted. Although in principle new entries could be insertion sorted into the current linear quadtree, Gargantini (1982) prefers a heapsort after the scanning phase, since that requires less work.

An implementation issue with linear arrays of locational codes is their storage: fixed spatial resolutions allow fixed size locational codes and hence efficient data structures and algorithms. Dynamic data structures for the locational code and even for each digit are possible, but require:

- a lot more memory,
- more complicated algorithms,
- memory management overhead such as garbage collection and heap compaction.

Lists of locational codes can be very large, with hundreds of thousands entries. Samet et al. (1984) have proposed a memory management system for quadtrees which allows very large quadtrees to be stored and manipulated. The linearly ordered nodes are stored on disc and only the portion of the tree for which there is room resides in memory if needed. A linked sequential B-tree whose index is a locational code is used to access the relevant virtual page. This concept is akin to a 'spatial working set' principle. Processing in spatial hierarchies is often quite localised, so that this strategy makes sense.

Each pixel is encoded in a weighted quaternary code, i.e. with digits 0,1,2,3 in base 4, where each successive digit represents the quadrant subdivision from which it originates. Similar encoding schemes exist for higher-dimensional rectangular (Gargantini 1982) and hexagonal hierarchies.

Compression of data structures is often useful, but memory requirements are not the only reason for using compression techniques. In many applications, hierarchies and locational codes can aid in processing information. In particular, it is often necessary to find the cell that is adjacent to the current one in a particular direction or to find all the cells adjacent to a given one. Adjacency operations are used in tracking processes and in clustering data and the locational code based algorithms are ideal candidates for this purpose. They can also be implemented in relatively simple hardware.

In clustering systems, geographic information systems and robot navigation problems, it is often necessary to find all nearest neighbours (obstacles in the robot case) of a given cell within a given radius. A straightforward, but inefficient method is to exhaustively generate the locational codes of all neighbouring coefficients within the given radius and to do a database lookup on each one. Again, it should be possible to exploit the local nature of this type of operation. A local nearest neighbour algorithm for pointer based quadtrees is described in (Samet 1990). A similar algorithm must exist for linear quadtrees (although it seems hitherto to be unpublished).

5. STATE OF THE WORK

The ideas presented in this paper are currently being expanded and implemented within the framework of the PhD-research project of the third author. The final result of this will be a program that is able to generate floorplans for buildings from initial specifications, without using precedents. The techniques described in this paper are further used in several other research co-operations.

6. REFERENCES

- Chisvin, L., Duckworth R.J. (1989) Content-Addressable and Associative Memory: Alternatives to the Ubiquitous RAM, *IEEE Computer*, July, pp. 52 - 64.
- Escher, M.C. (1989) *Grafiek en tekeningen*, Cordon Art, Baarn.
- Gargantini, I. (1982) An Effective Way To Represent Quadtrees, *Communications of the ACM*, Vol. 25, no. 12, Dec., pp. 905 - 910.
- Gibson, L., Lucas, D. (1982) Spatial Data Processing Using Generalised Balanced Ternary, *IEEE Pattern Recognition and Image Processing*, pp. 566 - 571.
- Grunbaum, B., Shephard, G.C. (1987) *Tilings and Patterns*, W. H. Freeman & Co, New York
- Jones, W.P., Dumas, S.P. (1986) The Spatial Metaphor for User Interfaces: Experimental Tests of Reference by Location versus Name, *ACM Transactions on Office Systems*, Vol. 4, nr. 1, Jan., pp. 42 - 63.
- Knuth, D.E. (1969) *The Art of Computer Programming*, Vol. 2 - Seminumerical Algorithms, Addison-Wesley, Amsterdam.
- Koenderink, J.J. (1990) *Solid Shape*, MIT Press, Cambridge, London.
- Lucas, D. (1979) A Multiplication in N-Space, *Proc. Americ. Math. Soc.* Vol. 74, nr. 1, April, pp. 1 - 8.

- Samet, H., Rosenfeld, A., Shaffer, C.A. (1984) Use of Hierarchical Data Structures in Geographic Information Systems, Proc. of the Int. Symp. on Spatial Data Handling, Zurich, August, Vol. 1, pp. 391 - 403.
- Samet, H. (1990) The Design and Analysis of Spatial Data Structures, Addison-Wesley, Amsterdam.
- Zandi-nia, A. (1992) TOPGENE: an AI Approach to a Design. PhD thesis, Fac. of Technical Mathematics and Informatics, Delft University of Technology.
- Zandi-nia, A., H. Koppelaar (1992) Design Optimisation with Neural Nets, International Journal for Intelligent Systems, Vol. 7, nr. 8, pp. 743 - 763.