Chapter 6
Some Schemes for the Modeling of n-Dimensional Polytopes

In this chapter we will describe some basic notions related to the Theory of Solid Modeling (section 6.1) and the Regularized Boolean Operations (section 6.2). In the section 6.3 three of the most known schemes for the modeling of solids will be briefly described: Boundary Representation, Cell Decomposition, Spatial Occupancy Enumeration and the Classical OctTrees. In section 6.4 the generalizations of these schemes for the Modeling of n-Dimensional Polytopes are presented and commented. In the section 6.4.2 the algorithm of Cohen & Hickey for the n-dimensional Simplexation (a way of n-Dimensional Cell Decomposition) of convex polytopes is described.

6.1 Solid Modeling

Solid Modeling is an area of wide development in several applications as the Computer Aided Design and Manufacturing (CAD/CAM), electronic prototypes and animation planning [Cardona, 01].

If a solid object can be modeled in a way that its geometry is appropriately captured, then it will be possible to apply, on such object, a range of useful operations. For example, it would be possible to determine if two objects interfere between them. Due to the need of modeling objects as solids, the development of a variety of specialized mechanisms to represent them has arisen.

[Requicha, 80] presents a set of formal criterions that every scheme for representing solids must have rigorously defined:

- **Domain:** The set of entities which are represented by the scheme. The domain's size must be enough to allow the representation of a useful set of objects, and therefore, it characterizes the scheme's power.
- **Completeness:** The representation can not be ambiguous. There are no doubts about what is represented. A representation must correspond to one and only one solid [Foley, 96].
- **Uniqueness:** A representation is unique if it can be used to codify a certain solid in just one way.
- **Validity:** A representation scheme must disable the creation of an invalid representation, or in other words, a representation that doesn't correspond to a solid. Additionally, the object must keep the closure under rotation, translation and other operations [Foley, 96]. In this way, the operations between valid solids must return valid solids.

Furthermore, [Requicha, 80] describes three informal properties (because they can not be rigorously defined) but with a great practical role in the schemes for representing solids:

- **Conciseness:** It refers to the representations' size. The concise representations should contain minimal redundant information.
- **Ease of Creation:** The users' tools for creating the designs should be simple. The concise representations are generally easier to create.

- **Efficacy in the Context of Applications:** The representation must allow the creation of efficient algorithms for computing the desired physical properties [Foley, 96].

The representation schemes for solid objects are frequently divided in two large categories (although not all the representations are completely inside in one of them): Boundary Representations and Spatial Partitioning Representations [Hearn, 95]. The most important aspects of these schemes are described in sections 6.3 and 6.4.

6.2 Regularized Boolean Operations

Independently of the objects' representation, it should be feasible to combine them to compose new objects. One of the most common methods to combine objects are the set theoretical Boolean operations, as the union, difference, intersection and exclusive OR. However, the application of an ordinary set theoretical Boolean operation on two solid objects doesn't necessarily produce a solid object. For example, the ordinary intersection between two cubes with a common vertex is a point.

Instead of using ordinary set theoretical Boolean operators, The **Regularized Boolean Operators** [Requicha, 77] will be used. Each regularized Boolean operator is defined in function of an ordinary operator in the following way:

$$A \; op^* \; B = Closure\,(Interior(A \; op \; B))$$

In such way we will have:

$A \cup^* B$ = Closure (Interior($A \cup B$))	Regularized Union
$A \cap^* B$ = Closure (Interior($A \cap B$))	Regularized Intersection
$A \otimes^* B$ = Closure (Interior($A \otimes B$))	Regularized Exclusive OR
$A -^* B$ = Closure (Interior($A - B$))	Regularized Difference

These operators are defined as the closure of the interior of the corresponding set theoretical Boolean operation [Aguilera, 98]. In this way, the regularized operations between solids always will generate solids. Recapturing the previous example, the regularized intersection between two cubes with a common vertex is the null object.

6.3 Some Schemes for the Modeling of Solids

6.3.1 Boundary Representations

In this model the solids are determined by the points that compose their boundary, because they separate the solid's interior points from the exterior points. The boundary is represented by a disjoint set of faces that can be planar or curved. When the faces are planar, each one is delimited by a ring perimeter of edges that intersect in the vertices. If the face has holes, it is delimited by one or more edges' internal rings. This type of representation is also known as Solid's Polyhedral Representation [Navazo, 86] or as B-Rep [Rossignac, 99].

The associated information to the components on a polyhedron's boundary (faces, edges and vertices) is composed by two parts:

- The Geometry: It considers the dimensions and localization in the space for each component. In this way, the points, edges and planes are defined.
- The Topology: It describes the connections between the elements. By this way, a point is identified as a vertex that limits a line that defines an edge. A ring of edges composes a polygon as the boundary of a surface that defines a face.

Independently of the representation that has been chosen for the boundary's geometry and topology, we must consider the restrictions for guaranteeing that an object is a valid solid. The characteristics that a solid's boundary must fulfill are [Foley, 96]:

- Each edge must contain only two vertices.
- Each edge must be shared by exactly two faces.
- At least three edges must be joined in each vertex.
- The faces can not penetrate between them.

Those solids that fulfill the previous properties satisfy the Euler's formulae, which expresses an invariable relation between the number of vertices, edges, and faces [Foley,96]:

$$V - E + F = 2$$

Where V is the vertices' number, E is the edges' number and F is the faces' number from the object. The Euler's generalized equation is applicable to those objects with holes, faces with holes or several components:

$$V - E + F - H = 2(C - G)$$

Where H is the number of holes in the faces, G is the number of holes that cross the object and C is the number of separate components (parts) of the object. The solids that fulfill the restrictions already mentioned and Euler's equation are those whose definition was presented in Section 2.1.1, that is to say, polyhedra whose edges and vertices are all manifold.

One of the most common representations requires four information listings or vectors. In [Argüelles, 02] each vector is defined in the following way:

- The first vector contains directly the geometric information of all the object's vertices, in other words, the coordinates (x_1, x_2, x_3) for each point.
- The second vector contains the topological information for the object's edges, that is to say, each one of its elements contains two pointers to the vertices' listing for the initial vertex and the final vertex of each edge from the solid.
- The third vector contains the topological information for the object's polygons. Each one of its elements contains pointers to the edges' listing for each one of the edges of each polygon from the solid, which are generally sorted counterclockwise.
- Finally, there is a fourth listing that contains the topological information for the object's faces. Each one of its elements contains pointers to the polygons' listing for each one of the polygons that compose each solid's face, from which generally the first one has the biggest surface and represents the external contour of a face, and the following, if there are, the internal contours (holes) of the face.

[Requicha, 00] describes that this type of representation is equivalent to a graph structure, called incidence graph [Hansen, 93], whose nodes belong to the faces, edges and vertices on the solid's boundary. The edges between the nodes express information about the connectivity. Together they constitute the combinatory structure (the topology) of the representation. The vertices' coordinates contain the metric information (the geometry) associated with the representation.

The boundary representations can be combined using the Boolean operators for the creation of new boundary representations [Requicha, 85]. The achievement of Boolean operations between solids, through a brute force procedure, are expensive in terms of the time for the computations. Because all the solid's faces must be evaluated against all the faces of other solid, and according to the specific operation, to decide which must be preserved [Argüelles, 01].

6.3.2 Spatial Partitioning Representations

In the Spatial Partitioning Representations, a solid is decomposed in a collection of attached solids, without intersections, and more primitive than the original solid, although they are not necessarily of the same kind [Foley, 96]. The primitives can vary in type, size, position, parameterization and orientation. The coverage of the objects' decomposition depends on how primitive the solids must be, to perform, in an easy way, the required operations.

6.3.2.1 Cell Decomposition

One of the most general representations for the spatial partitioning is the Cell Decomposition. The systems of cell decompositions define a set of primitive cells that are typically parameterized [Aguilera, 98]. A solid can be represented by decomposing it in cells with non intersecting interiors and by representing each cell in the decomposition. The Cell Decomposition representations provide convenient methods for the computing of certain topological properties for the objects. For example, to determine whether an object has just one component or it has holes [Requicha, 80].

An specific solids' decomposition is the tetrahedrization. The tetrahedrization of a polyhedron is its decomposition in tetrahedrons which must be either disjoint or to share a face, edge or vertex [Requicha, 80] (the 2D analogous of this scheme is a polygon's decomposition in triangles which define a triangulation).

6.3.2.2 Spatial Occupancy Enumeration

The solid is decomposed in identical cells each one positioned in a fixed and regular grid [Requicha, 80]. These cells are called voxels (volumetric cells). The most common type of cell is the cube (in [Herman, 98] is widely discussed the use of other types of cells). The representation of the space as a regular array of cubes is called a voxelization. Each cell can be represented by the coordinates of one of its points (the cell's centroid, the vertex with minimal coordinates, etc.) and its size is given by the grid's size. See in **Figure 6.1** an example of a 3D grid which can contain up to 8 voxels.

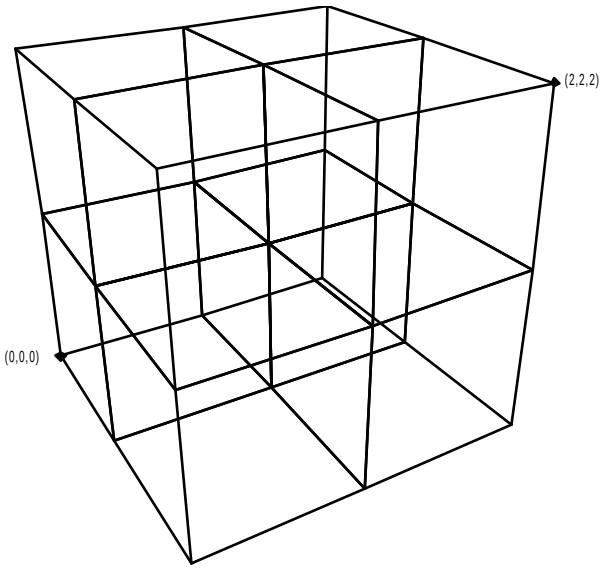


FIGURE 6.1
A 3D grid for positioning up to 8 unitary voxels
(own elaboration).

By representing an object through this scheme, only the presence or absence of each cell in the grid is controlled. The collection of cells can be efficiently codified as a three-dimensional array C_{ijk} of binary data. The array represents the coloration of each cube (cell) [Mäntylä, 88]:

- If $C_{ijk} = 1$, the black cube C_{ijk} represents a solid region of the space.
- If $C_{ijk} = 0$, the white cube C_{ijk} represents an empty region of the space.

There is no concept of "partial" occupation. Therefore, some objects only can be approximated. If the cells are cubes, then the only objects that can be exactly represented are the 3D-OPP's whose vertices coincide exactly with the grid [Aguilera, 98].

The design of algorithms for processing the objects represented by this scheme is direct. A basic example is the computing of the Boolean set operations. By using the

representation through a binary matrix, the algorithms just perform the operations between the bits for all the elements.

There are several applications that require the listing of the vertices for each voxel. Assuming that $C_{0,0,0}$ is the origin and the dimensions of each cubic cell are given by x_1Side, x_2Side and x_3Side, then the eight vertices for each voxel C_{ijk} are listed in the **Table 6.1**.

TABLE 6.1
Listing a voxel's eight vertices (see the text for details.
Taken from [Aguilera, 98]).

Vertex	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$
0	$i \cdot x_1Side$	$j \cdot x_2Side$	$k \cdot x_3Side$
1	$i \cdot x_1Side$	$j \cdot x_2Side$	$(k + 1) \cdot x_3Side$
2	$i \cdot x_1Side$	$(j + 1) \cdot x_2Side$	$k \cdot x_3Side$
3	$i \cdot x_1Side$	$(j + 1) \cdot x_2Side$	$(k + 1) \cdot x_3Side$
4	$(i + 1) \cdot x_1Side$	$j \cdot x_2Side$	$k \cdot x_3Side$
5	$(i + 1) \cdot x_1Side$	$j \cdot x_2Side$	$(k + 1) \cdot x_3Side$
6	$(i + 1) \cdot x_1Side$	$(j + 1) \cdot x_2Side$	$k \cdot x_3Side$
7	$(i + 1) \cdot x_1Side$	$(j + 1) \cdot x_2Side$	$(k + 1) \cdot x_3Side$

6.3.2.3 Classical OctTrees

It consists of a hierarchical tree structure generated by the recursive subdivision of a finite cubic universe. In this structure, each node is either a leaf or it has eight children. The tree divides the universe's space in cubes which can be inside or outside the object. The tree's root represents to the universe, a cube. This cube is divided in eight equal cubes denominated octants. Each octant is represented through one of the root's eight sorted

children. If an octant is partially inside the object, it is subdivided in another eight cubes. These new octants will be represented as children of the referred octant. The previous process is recursively repeated until there are obtained octants totally inside or outside of the object; or when the octants have an edge length sufficiently small (a minimal resolution) that represent the level of precision of the object [Argüelles, 02].

The size and location of a octant are determined by the level and the position of its associated node inside the tree. There are three types of nodes:

- Gray Nodes: The nodes associated with subdivided octants.
- Black Nodes: The nodes associated to octants totally inside the solid.
- White Nodes: The nodes associated to octants totally outside the solid.

One of the most common schemes for representing the classical OctTrees is the Tree Coding with Pointers. [Argüelles, 01] describes it in the following way:

- We have a tree whose nodes have 9 or 10 fields.
- One of the fields indicates the type of node (white, black or gray).
- Eight fields are pointers to the octants in which the given node is divided. If the node is a leaf then these eight pointers are nil.
- It is possible to have an additional field that is a pointer to the node from which the given node is an octant.

This model presents some of the most simple algorithms for performing the Boolean operations. The only restriction indicates that the initial cubical universe of the trees to

operate must have the same size and location. The topic related to the Boolean operations between OctTrees will be reconsidered in section 6.4.4.

6.4 Polytopes Modeling

The extension of the solid modeling schemes, by considering their application to spaces beyond the three-dimensional, have allowed the modeling of n-dimensional polytopes [Paoluzzi, 93]. In section 6.1 we mentioned the possibility of grouping the representation schemes for solid objects in two categories: Boundary Representations and Spatial Partitioning Representations. Since the following sections will deal with the extensions of the schemes (for the modeling of solids) previously commented (in section 6.3), is that we can propose two categories for grouping them:

- The n-Dimensional Boundary Representations: Where the polytopes are determined by the points that compose their boundary, i.e. the points that separate the polytopes's interior points from the exterior points.
- Hyperspatial Partitioning Representations: Where a polytope is decomposed in a collection of attached n-dimensional cells, without intersections, and more primitive than the original polytope. Inside this category we can find schemes as the n-Dimensional Cell Decomposition, the Hypervoxelization and the 2^n-trees (hyperoctress).

In the following sections we will describe the fundaments behind the nD boundary representations (6.4.1), the n-Dimensional "*Simplexation*" of Convex Polytopes (an specific cell decomposition; section 6.4.2), the hypervoxelizations (6.4.3) and, finally in section 6.4.4, the 2^n-trees.

6.4.1 The n-Dimensional Boundary Representations

A boundary model for a three-dimensional solid object is a description of the faces, edges and vertices that compose its boundary together with the information about the connectivity between those elements [Requicha, 80]. However, the boundary representations can be recursively applied not only to solids or surfaces or segments, but to hyperdimensional objects, or in other words, n-dimensional Polytopes [Hansen, 93].

A way to represent n-dimensional Polytopes through a boundary model is to consider n information listings or vectors. Each vector could be defined in the following way:

- The first vector contains the geometric information about all the polytope's vertices, that is to say, the coordinates (x_1, x_2, x_3, ..., x_n) of each one of its points.
- The second vector contains the topological information about the polytope's edges. Each one of its elements will contain two pointers to the vertices listing for the initial and final vertices of each edge of the polytope.
- The third vector contains the topological information about the polytope's polygons. Each one of its elements will contain pointers to the edges listing for each one of the edges in each one of the polytope's polygons.

- A k-th vector ($1 < k \leq n$) will contain the topological information about the cells Π_k from the polytope. Each element will have a set of pointers to the listing of the cells Π_{k-1} for each one of the Π_{k-1}'s in each one of the polytope's Π_k's.

For example, the representation for a 4D hypercube (n = 4) will require four listings:

- First Vector: It stores the coordinates (x_1, x_2, x_3, x_4) for its 16 vertices.
- Second Vector: It stores 32 elements, one for each edge and with two pointers to the vertices' listing.
- Third Vector: It stores 24 elements, one for each face and with four pointers to the edges' listing.
- Fourth Vector: It stores 8 elements, one for each volume and with six pointers to the faces' listing.

About the representations through graphs, [Hansen, 93] considers two possibilities:

- Extension of the concept of **Incidence Graph**: whose nodes belong to the cells Π_n, Π_{n-1},...,Π_1, Π_0 on the polytope's boundary. The edges between the graph's nodes express the information about the connectivity. Together, they constitute the combinatorial structure (the topology) of the representation. The vertices' n coordinates contain the metric information (the geometry) associated with the representation. In the **Figure 6.2** is presented the incidence graph for a 4D simplex.

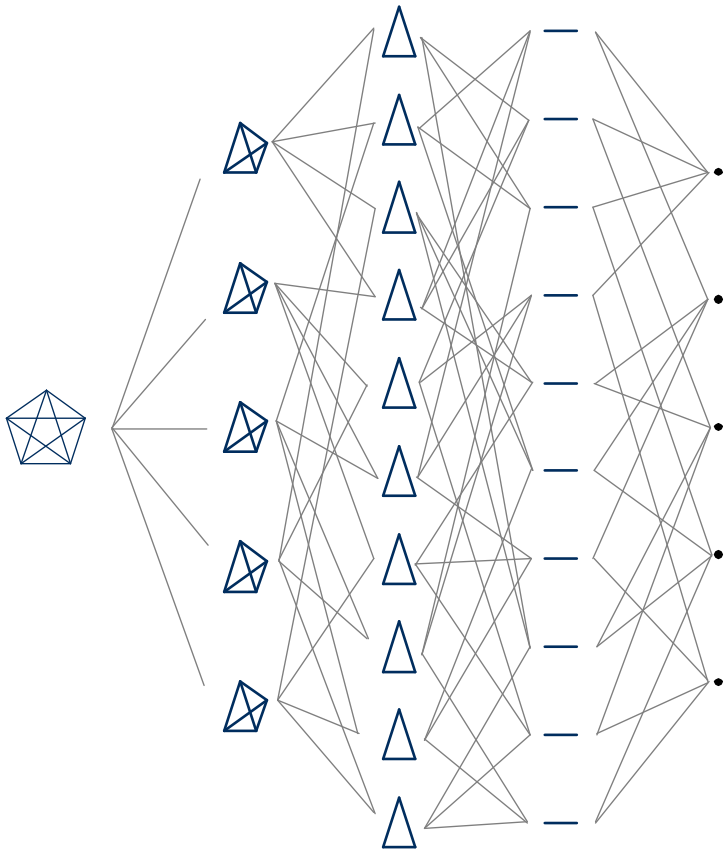


FIGURE 6.2
The incidence graph for the elements on the boundary of a 4D simplex (own elaboration).

- **Boundary Tree** (originally proposed by Putnam & Subrahmanyan): Where each node of the incidence graph is split into a component for each element that it bounds. An element (vertex, edge, etc.) will be represented several times inside the tree, one for each boundary that it belongs to. See **Figure 6.3** for a 4D simplex's boundary tree.

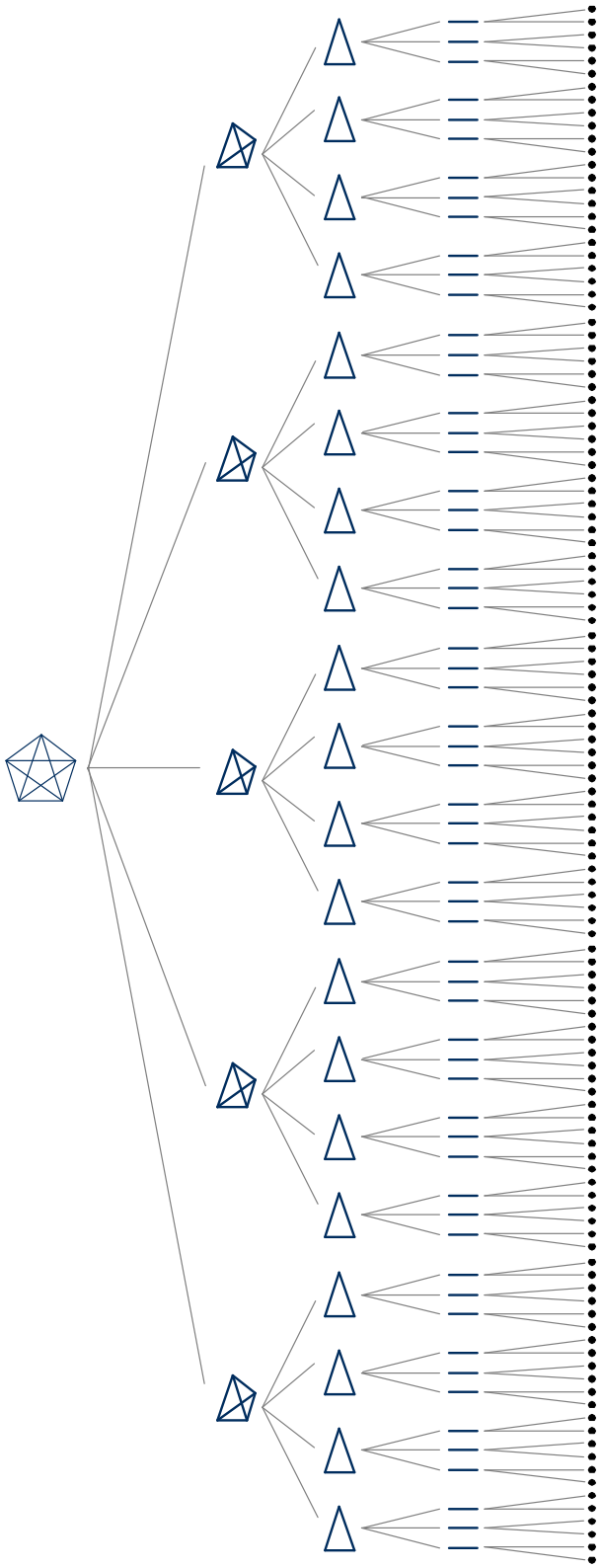


FIGURE 6.3
The boundary tree for a 4D simplex (own elaboration).

Independently of the representation to use, we must consider the hyperdimensional entities to be modeled. For example, the boundary models defined by [Hansen, 93] or [Gomes, 99] allow the representation of n-dimensional objects whose boundary can be orientable or not orientable, incomplete, or even without boundary. In our context, and in the way that was established in section 1.6.4, we will only consider the Polytopes' modeling. In section 6.3.1 was mentioned the use of Euler's formulae with the end of verifying whether a polyhedron representation was valid. Such recommendation can be applied inside the Polytopes' representation. The Euler's formulae that any simply connected 4D polytope (that is, without holes on its cells or completely crossing the polytope and with only one component) must fulfill will be [Coxeter, 63]:

$$N_3 - N_2 + N_1 - N_0 = 0$$

Where N_3 is the number of volumes (Π_3's), N_2 is the number of faces (Π_2's), N_1 is the number of edges (Π_1's) and N_0 is the number of vertices (Π_0's). Furthermore, there exists the generalization of the Euler's formulae for the simply connected n-dimensional polytopes [Sommerville, 58]:

$$\sum_{k=1}^{n}(-1)^{k-1}\cdot N_{n-k} = N_{n-1} - N_{n-2} + N_{n-3} - N_{n-4} + \ldots + (-1)^{n-1}N_0 = 1 - (-1)^n$$

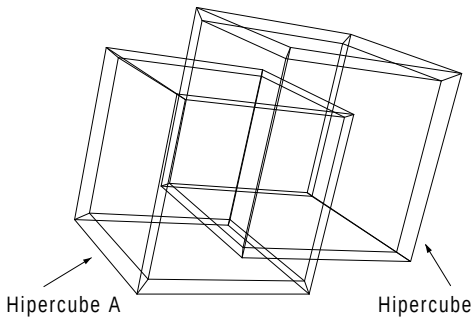
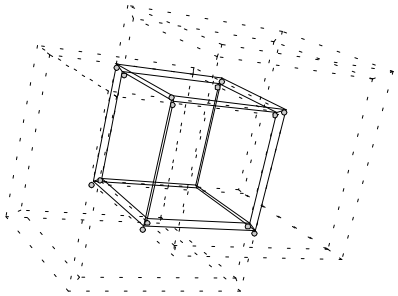
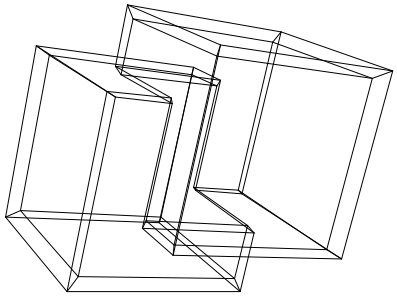
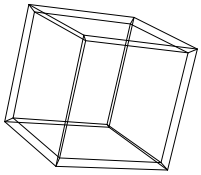
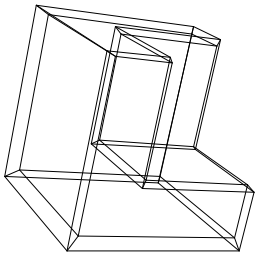
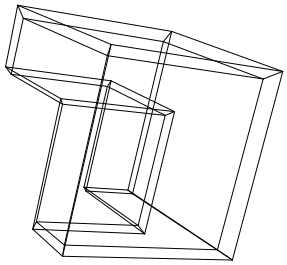
Where N_{n-1} is the number of (n-1)-dimensional cells (the Π_{n-1}'s), N_{n-2} is the number of (n-2)-dimensional cells (Π_{n-2}'s) and so forth until N_0 which is the number of vertices in the polytope.

A Boolean operation between two polytopes represented under a boundary representation scheme can be performed, according to the [Hansen, 93]'s procedure, through two main steps, that is, "cuts" and "sewings":

- The polytopes are subdivided (or "cut") in their intersecting boundary elements.
- Later on, the polytopes' subdivided elements are alternated and "sewn", after the consideration of which of them are preserved (according to the Boolean operation), to compose the new boundary or boundaries.

In the **Table 6.2** is shown an example of the Boolean operations between two 4D hypercubes (A and B) that have a common "corner" (1). The intersections between both hypercubes' volumes, faces and edges are computed. The boundary elements are "cut" according to their intersections and this way their boundaries are now subdivided. The intersections will describe a new hypercube embedded in both original hypercubes whose boundary is composed by their common parts (2). Depending on the Boolean operation to perform, we must decide what subdivided elements from the hypercubes must be considered and "sewn". In the table's cell 3 we have the union ($A \cup B$), in the cell 4 the intersection ($A \cap B$), in cell 5 the difference $A - B$ and finally, in the cell 6, B - A.

TABLE 6.2
The Boolean operations between two 4D hypercubes
(see the text for details; own elaboration).

1 	2 
3  $A \cup B$	4  $A \cap B$
5  $A - B$	6  $B - A$

6.4.2 The n-Dimensional Simplexation of Convex Polytopes

In this section we will describe the Cohen & Hickey's algorithm for the n-dimensional Simplexation of convex polytopes. In this scheme, a nD polytope is subdivided in a set of nD simplexes that not intersect between them. However, these same simple-

xes can share some of their boundary elements, that is, we can find vertex, edge, face, etc. adjacencies. In fact, a polygon's 2D simplexation is a triangulation; and a polyhedron's 3D simplexation is a tetrahedrization (section 6.3.2.1). In first place we will consider some definitions (6.4.2.1) and subsequently the algorithm will be described (6.4.2.2). The aspects concerning to formalizations and proofs are treated with the adequate detail in [Cohen,79].

6.4.2.1 Definitions

Cell: A cell is denoted by:

$$\Pi^{\text{index}}_{\text{dimensions}}$$

Then we have that:

Π^{i}_{0} denotes the i-th vertex of a polytope.

Π^{j}_{1} denotes the j-th edge of a polytope.

Π^{k}_{2} denotes the k-th face of a polytope.

$\vdots$

Π^{1}_{n} denotes to the polytope itself.

The function ψ: Let $\psi\left(\Pi^{i}_{d}\right)$ be the set of vertices from the i-th cell of d dimensions, i.e. the Π^{i}_{0}'s in Π^{i}_{d}.

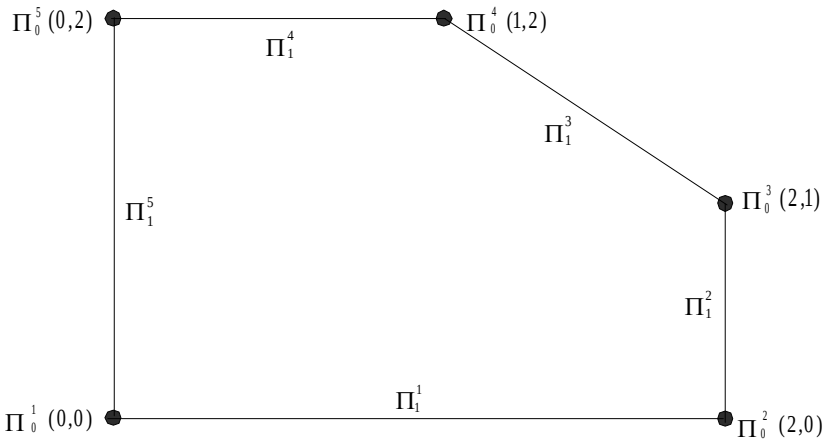


FIGURE 6.4

A polygon Π_2^1 and its cells Π_0^i and Π_1^j (own elaboration).

For example, by considering the polygon Π_2^1 presented in **Figure 6.4**, we have:

$$\psi\left(\Pi_2^1\right) = \left\{\Pi_0^1, \Pi_0^2, \Pi_0^3, \Pi_0^4, \Pi_0^5\right\}$$

$$\psi\left(\Pi_1^1\right) = \left\{\Pi_0^1, \Pi_0^2\right\}$$

$$\psi\left(\Pi_0^1\right) = \left\{\Pi_0^1\right\}$$

The function η: The mapping of a cell to a vertex is given by the function

$$\eta\left(\Pi_d^k\right) = \Pi_0^j \text{ where } j = \min\{i \mid \Pi_0^i \in \psi\left(\Pi_d^k\right), \text{ i.e. the vertex with the least index}\}$$

For example, in the polygon of **Figure 6.4** we have:

$$\eta\left(\Pi_2^1\right) = \min_i\left(\psi\left(\Pi_2^1\right) = \left\{\Pi_0^1, \Pi_0^2, \Pi_0^3, \Pi_0^4, \Pi_0^5\right\}\right) = \Pi_0^1$$

The function F_i: Let F_i be:

$$F_i = \psi\left(\Pi_{n-1}^i\right)$$

In other words, F_i is the set of vertices in a (n-1)-dimensional cell i.

The sets of vertices for the edges in the polygon of **Figure 6.4** will be:

$$F_1 = \psi\left(\Pi_1^1\right) = \left\{\Pi_0^1, \Pi_0^2\right\}$$

$$F_2 = \psi\left(\Pi_1^2\right) = \left\{\Pi_0^2, \Pi_0^3\right\}$$

$$F_3 = \psi\left(\Pi_1^3\right) = \left\{\Pi_0^3, \Pi_0^4\right\}$$

$$F_4 = \psi\left(\Pi_1^4\right) = \left\{\Pi_0^4, \Pi_0^5\right\}$$

$$F_5 = \psi\left(\Pi_1^5\right) = \left\{\Pi_0^1, \Pi_0^5\right\}$$

6.4.2.2 The Algorithm for the nD Simplexation of Convex Polytopes

The Cohen & Hickey's algorithm performs the simplexation of a polytope *p* by choosing any vertex $v \in p$ as an apex and connecting it with the (n-1)-dimensional simplexes that are the result of the simplexation of all the cells in *p* that do not contain *v*. Then, the pyramids with apex $\eta(\Pi_n)$ (remember that function η returns the vertex with the least index) and the bases among the cells Π_{n-1} with $\eta(\Pi_n) \notin \Pi_{n-1}$ will compose a dissection of the polytope [Büeler, 00].

The recursive application of this procedure on all the Π_{n-1} will form a set of decreasing cells $\Pi_n \supset \Pi_{n-1} \supset ... \supset \Pi_1 \supset \Pi_0$ such that $\eta(\Pi_i) \neq \eta(\Pi_j)$ for $1 \leq i, j \leq n$ and $i \neq j$. Then, the corresponding set of simplexes S=$\{\eta(\Pi_0), \ldots, \eta(\Pi_n)\}$ defines a simplexation of *p*. See in **Figure 6.5** an example of this process related to the composition of a tetrahedron inside a cube. In **Figure 6.5.a** is shown the selection of each one of the four required $\eta(\Pi_3)$, $\eta(\Pi_2)$, $\eta(\Pi_1)$ and $\eta(\Pi_0)$. When we reach the basic case, these four vertices will compose the set S that contains the vertices of a tetrahedron, that is, S=$\{\eta(\Pi_3), \eta(\Pi_2), \eta(\Pi_1), \eta(\Pi_0)\}$ (**Figure 6.5.b**).

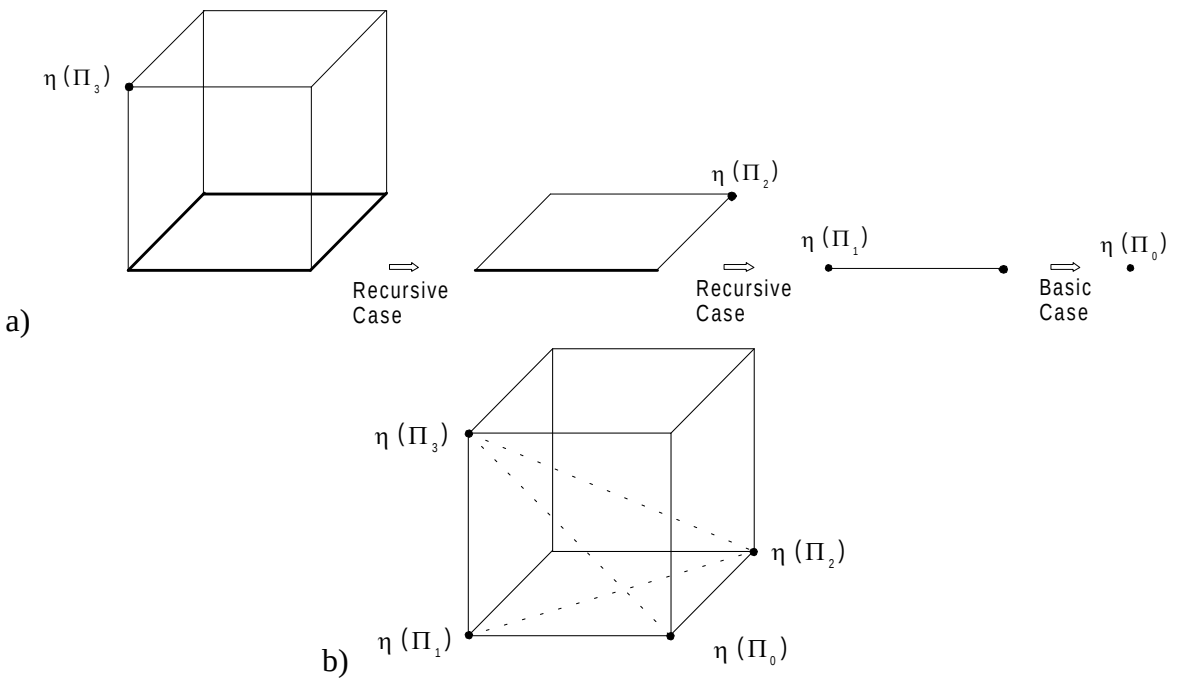


FIGURE 6.5

Forming a tetrahedron inside a cube (see text for details; own elaboration).

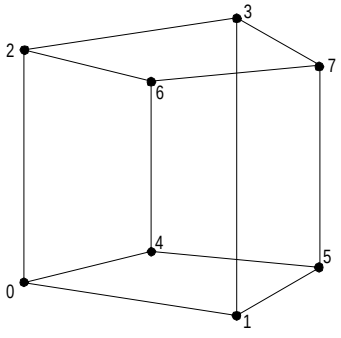
The implementation of this recursive procedure requires that the cells Π_{n-1} be represented as sets of vertices, i.e. through function F_i. By starting from this representation we pass from a cell Π_k to Π_{k-1} by intersecting the set of vertices in Π_k with the cells Π_{n-1} from *p* that not contain the vertex $\eta(\Pi_k)$. To avoid the multiple generation of a cell we maintain a list that contains all the cells Π_k earlier generated; only the cells not included in the list are processed [Büeler, 00].

The algorithm will require initially three input parameters:

- d: Number of dimensions.
- last: A set that contains all the vertices from the polytope, i.e. $\psi\left(\Pi_n^1\right)$.
- S: The set that contains the vertices of the nD simplex in construction. In the algorithm's main call S={ Π_0^1 }.

```
simplexation (d, last, S)
{
        // m: the number of (n-1)-dimensional cells in the original polytope.
        // F_k: The set of vertices of each (n-1)-dimensional cell in the original polytope.
        // L: A list of sets.
        If (d > 0)
        {
                L = {{ }}
                For k = 1 until m
                {
                        I = last ∩ F_k  // I is a candidate set to represent ψ(Π_d^j).
                        If ( I ∉ L ) // It is evaluated if the set was not earlier obtained.
                        {
                                L = {I} ∪ L // The set ψ(Π_d^j) is added to the list L.
                                If ( η(I) ∉ S ) /* Verifying if vertex η(I) is not contained in
                                                        current simplex. */
                                        simplexation (d-1, I, η(I) ∪ S )
                        }
                }
        }
        else
                S contains the vertices of a n-dimensional simplex.
}
```

For example, by applying this algorithm on a cube with the coordinates presented in **Figure 6.6**, we obtained its simplexation (i.e. a tetrahedrization) which is composed by six tetrahedrons whose respective vertices are shown in **Figure 6.7**. In **Figure 6.8** is shown the graphical result.



Vertex	X_1	X_2	X_3
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

FIGURE 6.6
A cube and its coordinates (own elaboration).

S_i	V_0	V_1	V_2	V_3
1	0	4	5	7
2	0	4	6	7
3	0	1	5	7
4	0	1	3	7
5	0	2	6	7
6	0	2	3	7

FIGURE 6.7
The vertices of the 6 tetrahedrons that compose the 3D simplexation of a cube (from Figure 6.6; own elaboration).

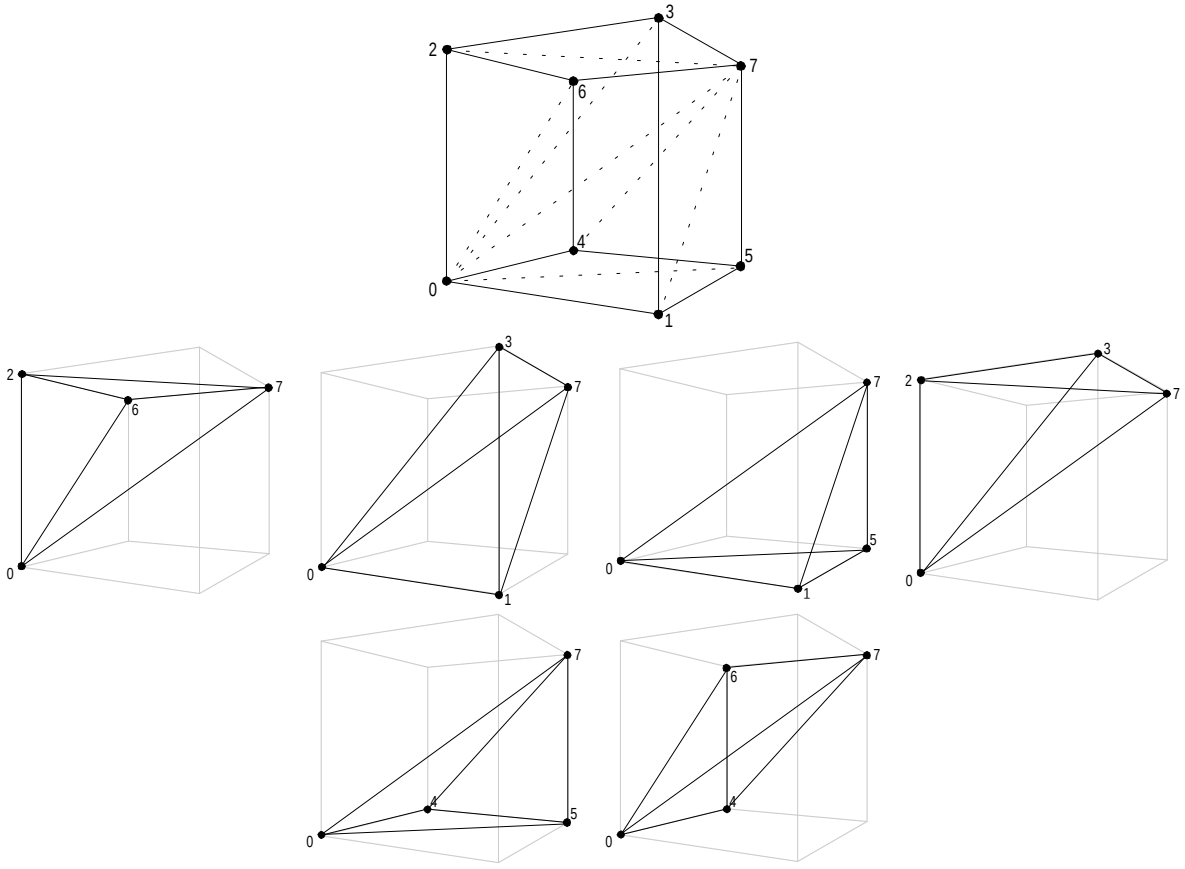
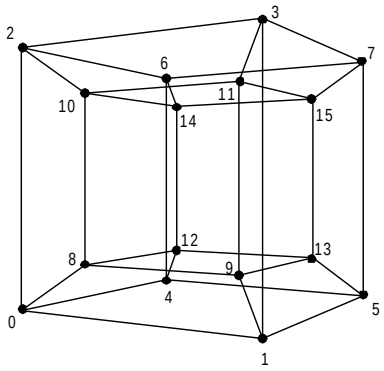


FIGURE 6.8
The resultant six tetrahedrons from the "triangulation" of a cube through the Cohen & Hickey's algorithm (own elaboration).

In another example, by applying the Cohen & Hickey's algorithm on a 4D hypercube with the coordinates presented in **Figure 6.9**, we obtained its 4D simplexation which is composed by 24 simplexes whose respective vertices are shown in **Figure 6.10**. In **Figure 6.11** is shown the graphical result.



Vertex	X_1	X_2	X_3	X_4	Vertex	X_1	X_2	X_3	X_4
0	0	0	0	0	8	0	0	0	1
1	1	0	0	0	9	1	0	0	1
2	0	1	0	0	10	0	1	0	1
3	1	1	0	0	11	1	1	0	1
4	0	0	1	0	12	0	0	1	1
5	1	0	1	0	13	1	0	1	1
6	0	1	1	0	14	0	1	1	1
7	1	1	1	0	15	1	1	1	1

FIGURE 6.9
A cube and its coordinates (own elaboration).

S_i	V_0	V_1	V_2	V_3	V_4
1	0	8	10	11	15
2	0	8	10	14	15
3	0	8	9	11	15
4	0	8	9	13	15
5	0	8	12	14	15
6	0	8	12	13	15
7	0	2	10	11	15
8	0	2	10	14	15
9	0	2	3	11	15
10	0	2	3	7	15
11	0	2	6	14	15
12	0	2	6	7	15
13	0	1	9	11	15
14	0	1	9	13	15
15	0	1	3	11	15
16	0	1	3	7	15
17	0	1	5	13	15
18	0	1	5	7	15
19	0	4	12	14	15
20	0	4	12	13	15
21	0	4	6	14	15
22	0	4	6	7	15
23	0	4	5	13	15
24	0	4	5	7	15

FIGURE 6.10
The vertices of the 24 simplexes that compose the 4D simplexation of a 4D hypercube (from Figure 6.9; own elaboration).

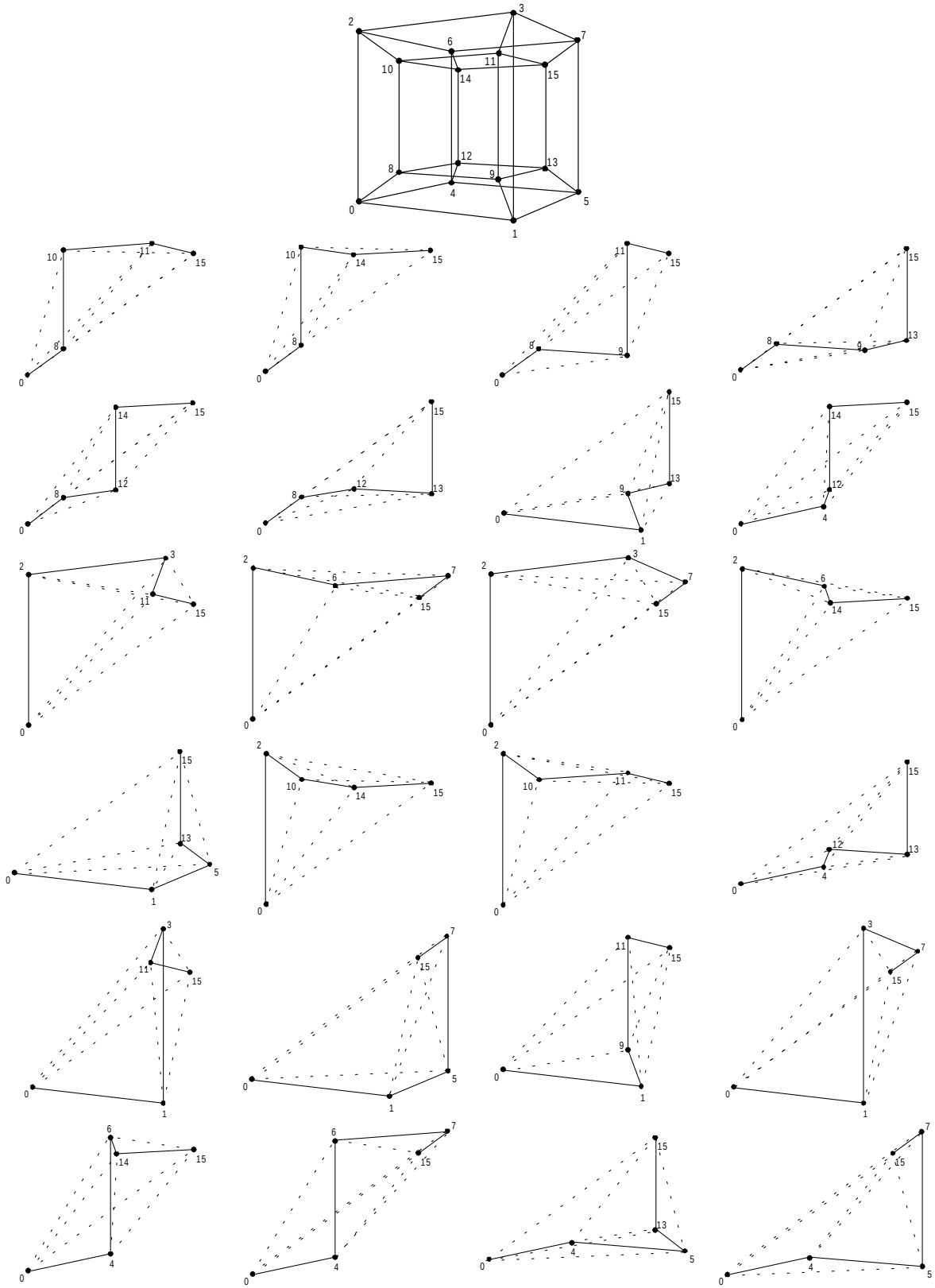


FIGURE 6.11

The resultant 24 simplexes from the simplexation of a 4D hypercube (from Figure 6.9) through the Cohen & Hickey's algorithm (own elaboration).

In fact, [Büeler, 00] points out that the time complexity for the simplexation of a n-dimensional hypercube through the Cohen & Hickey's algorithm is $O(n^3 n!)$.

6.4.3 Hypervoxelization

The representation of a polytope through a scheme of Hyperspatial Occupancy Enumeration is essentially a list of identical hyperspatial cells occupied by the polytope. An specific type of cells, called hypervoxels [Jonas, 95] are hyper-boxes of a fixed size that lie in a fixed grid in the n-dimensional space. [Jonas, 95] defines two kinds of hypervoxels:

- Centered Hypervoxel: a n-dimensional hyper-box whose dimensions are given by x_1Side, x_2Side, ..., x_nSide and it is represented by the coordinates of its centroid.
- Shifted Hypervoxel: whose characteristics are same that those for the centered hypervoxel, except that its representation is given by some of its 2^n vertices.

By instantiation, we know that a 2D hypervoxel is a pixel while a 3D hypervoxel is a voxel; the term *rexel* is suggested for referencing a 4D hypervoxel [Jonas,95]. See in the **Figure 6.12** an example of a 4D grid which can contain up to 16 *rexels* or 4D hypervoxels.

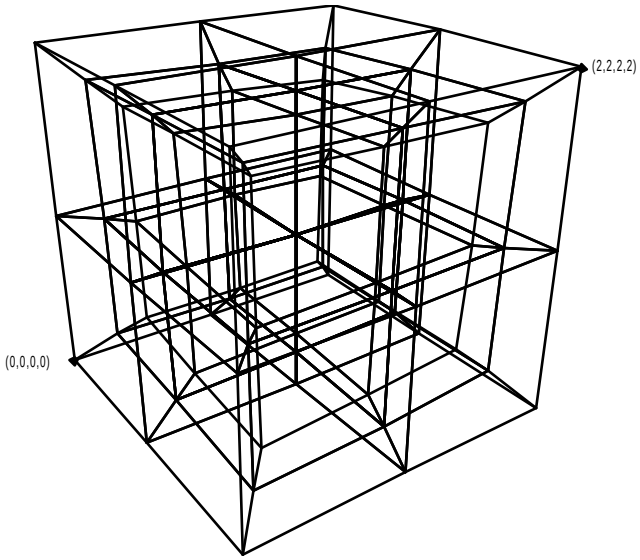


FIGURE 6.12
A 4D grid for positioning up to 16 unitary *rexels* (own elaboration).

The collection of hyperboxes can be codified as a n-dimensional array $C_{x_1,x_2,...,x_n}$ of binary data. The array will represent the coloration of each hypervoxel:

- If $C_{x_1,x_2,...,x_n} = 1$, the black hypervoxel $C_{x_1,x_2,...,x_n}$ represents an occupied region from the n-dimensional space.
- If $C_{x_1,x_2,...,x_n} = 0$, the white hypervoxel $C_{x_1,x_2,...,x_n}$ represents an empty region from the n-dimensional space.

By using the representation through a binary matrix, the computation of the Boolean set operations just control the operations between bits for all the elements. Let C^1 and C^2 be two n-dimensional grids of hypervoxels, then the Boolean operations between their respective cells, $C^1_{x_1,...,x_n}$ *op* $C^2_{x_1,...,x_n}$, are defined as shown in the **Figure 6.13**.

$C^1_{x_1,...,x_n} \cup C^2_{x_1,...,x_n}$	1	0
1	1	1
0	1	0

$C^1_{x_1,...,x_n} \cap C^2_{x_1,...,x_n}$	1	0
1	1	0
0	0	0

$C^1_{x_1,...,x_n} \otimes C^2_{x_1,...,x_n}$	1	0
1	0	1
0	1	0

$C^1_{x_1,...,x_n} - C^2_{x_1,...,x_n}$	1	0
1	0	1
0	0	0

$C_{x_1,...,x_n}$	$\overline{C_{x_1,...,x_n}}$
1	0
0	1

FIGURE 6.13
Boolean operations between two hypervoxels' grids C^1 and C^2
(own elaboration).

In the **Figure 6.14** (a and b) are presented two 5D-OPP's embedded in a 5D hypercubic universe (in **Figure 3.3** can be observed the central projection of a 5D hypercubes as a 4D hypercube inside another 4D hypercube). In the **Figures 6.14.c** and **d** are shown their representations through 5D hypervoxels (both 5D-OPP's required 24 cells).

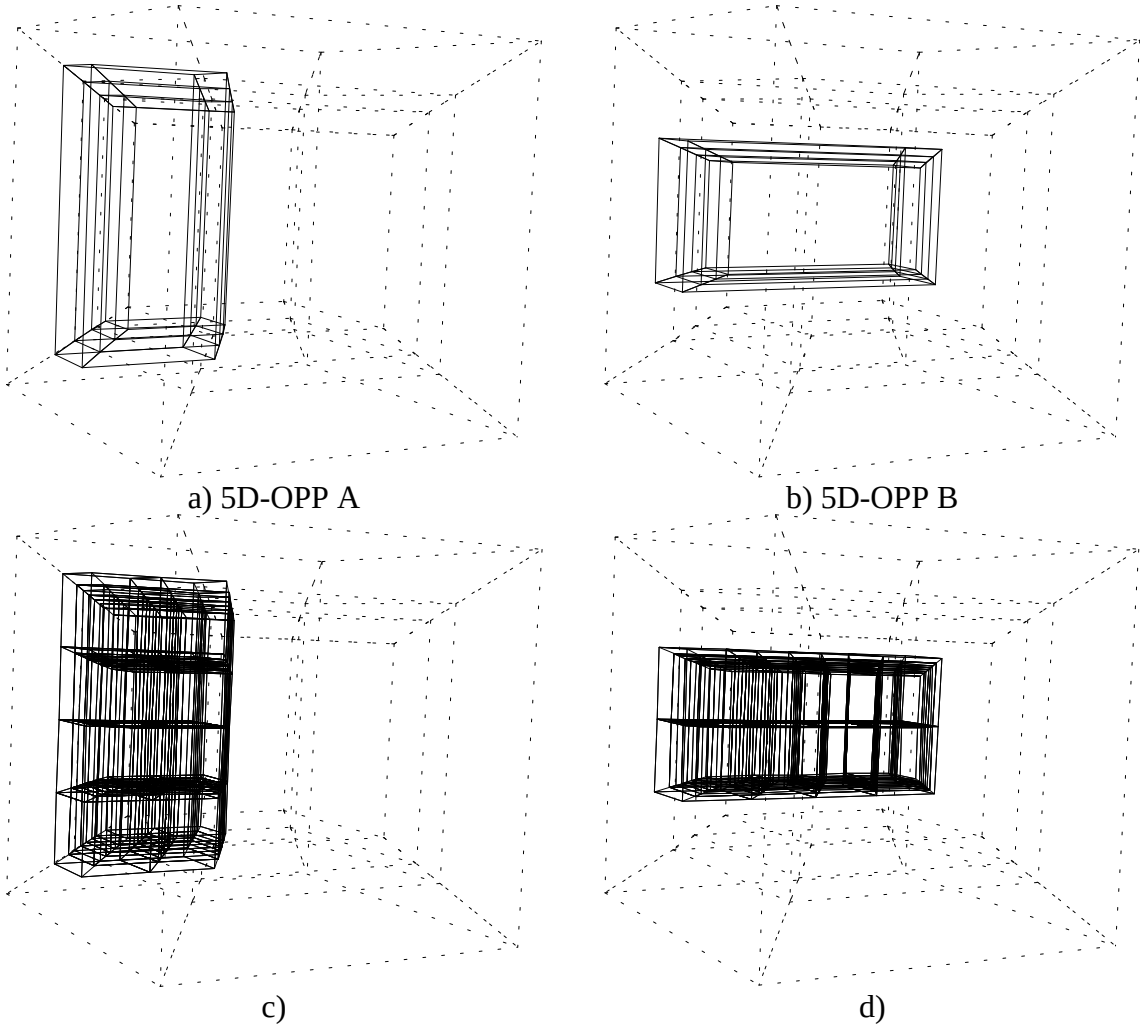


FIGURE 6.14
Two 5D-OPP's (a and b) embedded in a 5D hypercubic universe (dotted lines) and their representation through hypervoxels (c and d, respectively; own elaboration).

In **Figure 6.15** are shown the Boolean operations between the 5D-OPP's from **Figure 6.14**. For each case, union (a and b), intersection (c and d), difference *A* - *B* (e and f) and difference *B* - *A* (g and h), are shown both their representation through 5D hypervoxels and their central projection. The resultant polytope from the union (**Figure 6.15.a**) has 36 hypervoxels; the resultant 5D-OPP's from the intersection, difference *A* - *B* and the difference *B* - *A* (**Figures 6.15.c**, **e** and **g**, respectively) have 12 hypervoxels.

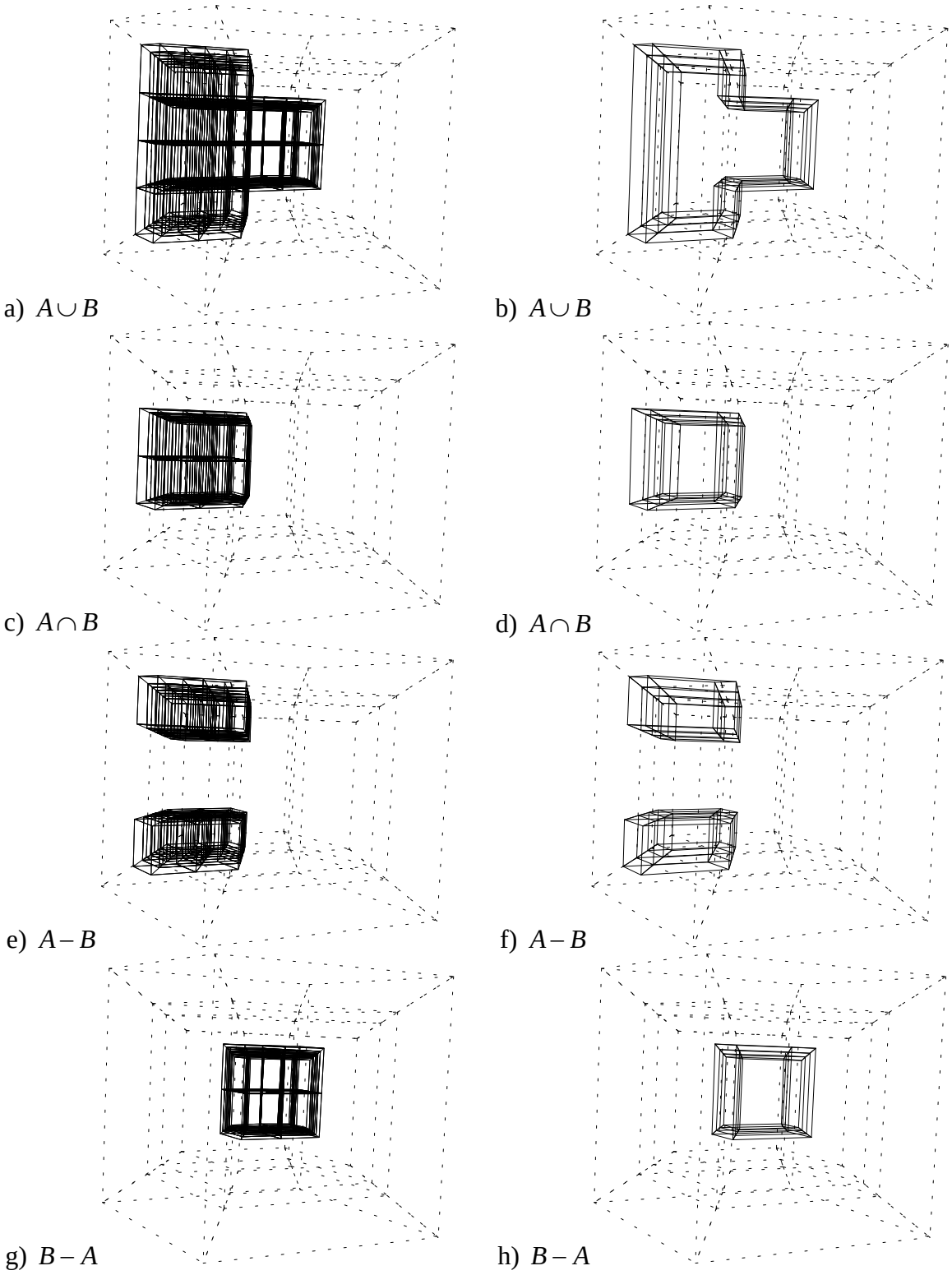


FIGURE 6.15

Results of the Boolean Operations between the two 5D-OPP's from Figure 6.14 (the left column show the representation through hypervoxels, the right column show their central projection; own elaboration).

We will extend the procedure commented in section 6.3.2.2 in order to list the 2^n vertices of a hypervoxel. Consider a n-dimensional grid where $C_{\underbrace{0,0,0,...,0}_{n}}$ is the origin and the dimensions of each hypercubic cell are given by x_1Side, x_2Side, x_3Side, ..., x_nSide. In section 2.2.1.4 was presented the general set of the coordinates for a n-dimensional unitary hypercube:

$$(\underbrace{0,0,...0,0}_{n}), (\underbrace{1}_{1},\underbrace{0,...,0,0}_{n-1}), ..., (\underbrace{1,...,1}_{i},\underbrace{0,...,0}_{n-i}), ..., (\underbrace{1,1,...,1}_{n-1},\underbrace{0}_{1}), (\underbrace{1,1,...,1,1}_{n}) =$$

$$(1^0,0^n),\ (1^1,0^{n-1}),\ ...,\ (1^i,0^{n-i}),\ ...,\ (1^{n-1},0^1),\ (1^n,0^0)$$

Where the coordinates must be permuted according to the distribution:

$$C\binom{n}{0},\ C\binom{n}{1},\ ...,\ C\binom{n}{i},\ ...,\ C\binom{n}{n-1},\ C\binom{n}{n}$$

Where $C\binom{n}{i} = \frac{n!}{i!(n-i)!}$ defines the number of those coordinates that have i ones and n-i zeros. Such set will be adapted with the end of obtaining the set of coordinates for a n-dimensional hypervoxel $C_{x_1,...,x_n}$. Therefore, we only need to apply the translation $(\underbrace{i, j, k,...}_{n})$ and the scaling (x_1Side, x_2Side, x_3Side, ..., x_nSide) to the general coordinates for obtaining the set of specific coordinates. For example, in **Table 6.3** is presented the listing of the 16 vertices from a *rexel* $C_{i,j,k,\ell}$.

TABLE 6.3

Listing a rexel's sixteen vertices (see text for details; own elaboration).

Vertex	X_1	X_2	X_3	X_4
0	$i \cdot x_1Side$	$j \cdot x_2Side$	$k \cdot x_3Side$	$\ell \cdot x_4Side$
1	$i \cdot x_1Side$	$j \cdot x_2Side$	$k \cdot x_3Side$	$(\ell+1) \cdot x_4Side$
2	$i \cdot x_1Side$	$j \cdot x_2Side$	$(k+1) \cdot x_3Side$	$\ell \cdot x_4Side$
3	$i \cdot x_1Side$	$j \cdot x_2Side$	$(k+1) \cdot x_3Side$	$(\ell+1) \cdot x_4Side$
4	$i \cdot x_1Side$	$(j+1) \cdot x_2Side$	$k \cdot x_3Side$	$\ell \cdot x_4Side$
5	$i \cdot x_1Side$	$(j+1) \cdot x_2Side$	$k \cdot x_3Side$	$(\ell+1) \cdot x_4Side$
6	$i \cdot x_1Side$	$(j+1) \cdot x_2Side$	$(k+1) \cdot x_3Side$	$\ell \cdot x_4Side$
7	$i \cdot x_1Side$	$(j+1) \cdot x_2Side$	$(k+1) \cdot x_3Side$	$(\ell+1) \cdot x_4Side$
8	$(i+1) \cdot x_1Side$	$j \cdot x_2Side$	$k \cdot x_3Side$	$\ell \cdot x_4Side$
9	$(i+1) \cdot x_1Side$	$j \cdot x_2Side$	$k \cdot x_3Side$	$(\ell+1) \cdot x_4Side$
10	$(i+1) \cdot x_1Side$	$j \cdot x_2Side$	$(k+1) \cdot x_3Side$	$\ell \cdot x_4Side$
11	$(i+1) \cdot x_1Side$	$j \cdot x_2Side$	$(k+1) \cdot x_3Side$	$(\ell+1) \cdot x_4Side$
12	$(i+1) \cdot x_1Side$	$(j+1) \cdot x_2Side$	$k \cdot x_3Side$	$\ell \cdot x_4Side$
13	$(i+1) \cdot x_1Side$	$(j+1) \cdot x_2Side$	$k \cdot x_3Side$	$(\ell+1) \cdot x_4Side$
14	$(i+1) \cdot x_1Side$	$(j+1) \cdot x_2Side$	$(k+1) \cdot x_3Side$	$\ell \cdot x_4Side$
15	$(i+1) \cdot x_1Side$	$(j+1) \cdot x_2Side$	$(k+1) \cdot x_3Side$	$(\ell+1) \cdot x_4Side$

6.4.4 The HexTrees and 2^n-trees (Hyperoctrees)

As commented in section 6.3.2.3, the octtrees are composed starting from the recursive subdivision of a 3D cubic space in eight octants until each octant is reduced (arbitrary) in the possible simplest way. The consideration of this method of recursive subdivision lead us to the definition of a *hextree* of 16 hyper-octants. Moreover, the generalization of this hierarchical tree structure lead us to the recursive division of a n-dimensional space in 2^n hyper-octants which is called a 2^n-tree or hyperoctree [Srihari,83].

Such as the 2D and 3D cases, the 2^n-tree will have three types of nodes:

- Gray Nodes: The nodes that correspond to hyper-octants not completely full nor not completely empty. These nodes must be subdivided.

- Black Nodes: The nodes that correspond to hyper-octants completely occupied.
- White Nodes: The nodes that correspond to hyper-octants completely empty.

The root node from the 2^n-tree corresponds to the entire n-dimensional space, that is, a n-dimensional hypercube that contains (or encloses) a n-dimensional polytope. The conceptual procedure for the building of the tree is the same that is applied to the quadtrees or octtrees: If a cell is full or empty, then it must be marked as black or white, respectively; otherwise it must be marked as gray and subdivide it recursively [Requicha, 00].

By representing a 2^n-tree through a Tree Codification with Pointers we would have to consider the following characteristics:

- Each node of the tree will contain $2^n + 1$ or $2^n + 2$ fields.
- One of the fields will indicate the kind of node (white, black or gray).
- 2^n fields will be pointers to the hyper-octants in which the given node is divided. If the node is a leaf then these 2^n pointers will be nil.
- It is possible to have an additional field that is a pointer to the node from which the given node is an hyper-octant.

The achievement of Boolean operations between two 2^n-trees follows the same procedures that are applicable to quadtrees or octtrees [Srihari, 83]. Only a consideration must be observed, that is, the initial nD hypercubic universe from both trees to operate must have the same size and location.

The achievement of the complement operation consists in traverse the codification of a 2^n-tree changing the white nodes by black nodes and vice versa. Now, the procedure for the computing of the union or intersection T_3 between two trees T_1 and T_2 will be described (specifically the union's case):

1. A parallel descending traverse in both trees is performed.
2. Each corresponding homologous pair of nodes (that is, with the same size and location) is examined. If some of the nodes in the pair is black, then it is added a corresponding black node in T_3.
3. If one of the nodes in the pair is white, then it is created in T_3 the corresponding node with the value of the other node in the pair.
4. If both nodes in the pair are gray, then it is added a gray node in T_3 and the algorithm is recursively applied to the pair's sons. In this case the sons of the new node in T_3 must be inspected after the application of the algorithm. If all are black, then they are eliminated and its father in T_3 changes from gray to black.

The intersection between two trees follows the same procedure before described only considering the criteria to apply according to the corresponding pair of nodes. In the **Figure 6.16** are presented the results of the operations of union and intersection between two nodes and the complement for a node.

$T_1 \cap T_2$	B	W	G
B	B	W	G
W	W	W	W
G	G	W	G^*

$T_1 \cup T_2$	B	W	G
B	B	B	B
W	B	W	G
G	B	G	G^*

T	$\overline{T}$
B	W
W	B
G	G^*

FIGURE 6.16

The operations of union, intersection and complement for the nodes from 2^n-trees (B: Black node, W: White node, G: Gray node, G*: recursive case; own elaboration).

Other operations such as the Difference or Exclusive Or can be easily derived through the three operations before described (union, intersection and complement). By this way, the difference can be determined starting from the well known expression:

$$A - B = A \cap \overline{B}$$

While the Exclusive OR is calculated through:

$$A \otimes B = (A \cup B) - (A \cap B)$$

In **Figure 6.17** are presented the results of the Difference and Exclusive OR operations between two nodes.

T_1 - T_2	B	W	G
B	W	W	W
W	W	B	$\overline{G}$
G	W	G	G*

$T_1 \otimes T_2$	B	W	G
B	B	W	G
W	W	B	$\overline{G}$
G	$\overline{G}$	G	G*

FIGURE 6.17

The operations of difference and Exclusive OR for the nodes from 2^n-trees (B: Black node, W: White node, G: Gray node, $\overline{G}$: Gray node's complement, G*: recursive case; own elaboration).