Prototype Shape Modeling with a Design Language

*Alberto Paoluzzi ° Valerio Pascucci ° Claudio Sansoni**

°Dip. di Disc. Scientifiche: sez. Informatica, Terza Università
Via C. Segre, 2, 00146 Roma, Italy

*Dip. di Prog. e Scienze dell'Architettura, Terza Università
Via della Madonna dei Monti 40, 00186 Roma, Italy

Abstract A programming approach to the rapid prototyping of architectural design is discussed in this paper. This is done with particular reference to the early steps of design development, where a number of preliminary design alternatives should be generated and evaluated. At this purpose we show that the generation of the 3D shape of each design alternative can be automated starting from the 2D layout of plans, sections and elevations. Each such geometric object can be symbolically defined with few lines of code using design variables and constraint operators. The 3D models generated by evaluation of program scripts may then be used as input to standard engineering evaluation methods concerning costs, heat exchanges and structural behaviour.

1. Introduction

The aim of this paper is to show the capability of the geometric language PLaSM (Programming Language for Symbolic Modeling) to aid the designer in the early phases of the design process, focusing on the peculiarities of architectural shape generation and manipulation.

Most of the current CAAD systems are mainly based on drawing tools integrated with database functionalities, and constitute effective aids in the detailed design definition, but are not still useful in the early steps of the design development. With the introduction of variational geometry new perspectives appear, since it become possible to experiment alternatives in the shape definition. This approach is prevalent in mechanical engineering since it well exploits the degrees of freedom really necessary in the generation of mechanical parts.

A CAAD system useful in the early design should allow the designer to explore the shape alternatives in a domain as wide as possible. Even a variational approach does not provide such a capability since it allows only to modify some geometric characteristics of an object with a basically fixed topology. Also, the constraint-satisfaction approach may be convenient when the design process is mainly bottom-up, as often happens in mechanical design, but may be very slow and costly when the project must be developed in a top-down manner. Conversely, a top-down approach is particularly useful in the architectural design, where is often necessary to change some early design decisions, in order to test the alternatives in a trial and error process.

In advanced CAAD systems one of main goals is to build advanced user interfaces to make easier the generation of geometric models. The techniques and methods on how the architect should define the geometric model of the shape are established in advance by the designer of the CAAD system. So the system designer often imposes a foreign way of work to the user of the system. This approach involves a remarkable degree of rigidity when the system is easy to use and the user interface becomes more friendly. Conversely, a geometry-oriented programming interface gives the advanced user the widest freedom and flexibility.

PLaSM is a design language, developed by the CAD group at “La Sapienza” in early nineties and currently maintained at the “Terza” University of Rome, that implements a programming approach to the geometric design.

As we show in the paper, a geometric programming approach to shape generation allows to describe a design as a set of symbolic definitions rather than to directly give the geometry of the shape. Any definition may depend on other definitions, i.e. on the current status of the design knowledge basis. So, the evaluation of a single "generating expression" in the language may result in many different instances of the same shape prototype. In other words, PLaSM allows to describe a whole class of objects by representing the set of their common characteristics. Notice that each design solution can be characterized by a very different topological structure, conversely to the standard variational approach.

In this paper, starting from a "sketch symbolic description" of some exemplar cases, it is shown how the degrees of freedom provided by such a programming approach allow the designer to investigate the alternatives in the working out of the shape. The development of some simple examples in the 2D domain and of more complex 3D models is aimed to show the expressive power of such an approach that, taking a dimension-independent viewpoint, can homogeneously manipulate geometric objects of different dimensionality.

The paper is organized as follows. In Section 2 some aspects of computer aided shape generation and morphology management are analyzed with special reference to design in architecture and to the methods and tools previously developed in this area. In Section 3 some methodology for the programming approach with a functional language is quickly recalled. In Section 4 the programming tools actually used in the following case study are presented. In Section 5 the investigation of alternatives in the working out of the design shape is discussed, with reference to the Wislocki house by Robert Venturi.

2. Morphology management in early design stages

The architectural design of a building results from a complex synthesis between the satisfaction of functional constraints and the need of a pleasant aesthetics. So, the early steps of the design process may start both from the activity requirement analysis as well as from a global conception of the shape, to undergo to several checks about qualities and organization of spaces. The first approach requires to control the adjacency relationships between elementary spaces, the second one is mainly devoted to the control of the shape of the designed building as a whole. In any case, the geometric aspects of space planning are the basic ingredients of any design decision. Actually, both approaches are contemporaneously considered by the architect and are cyclically taken into account until a satisfying solution is reached for both.

The geometric model of a building is built in several ways according to the design stage. In the early phases of work we deal with both the global envelope and the organization of internal spaces. When the design is generally defined we can study in definitive manner the character of the external and internal partitions. When the design becomes more detailed the geometric shape of the elements of the building fabric become meaningful. It is possible to single out several approaches to the early modeling of shape. Three different methods to deal with the geometric description according to the early levels of design probing follow.

1. Modeling the external envelope, i.e. the shell of the building. This approach is particularly useful in urban design as well as for visual impact analysis. At this purpose it is possible to use either a surface or a solid modeler. In the last case, because we are not interested to the internal space of the building, we can model it as a set of full solid blocks.
2. Modeling the partitions. In this case the goal is to generate an accurate model of the partitions. This model is useful in the next steps of the design process when it is important to make several checks and analysis on the model. Besides the usual visual controls, such an approach allows to verify quite carefully the amount of money necessary for the construction as well as the behaviour of the structures and the environmental efficiency of the building.

3. Modeling the envelope together with the internal spaces. This approach is the newest indeed in such a case the goal is to model the "empty spaces". This approach is particularly suited in the early design. It may be useful to make use of a solid modeler based on cell decompositions, possibly using cells with incomplete boundaries. It is also useful to be able to utilize at the same time both 2D and 3D geometric objects, according to the standard architectural design practice.

The last modeling approach is suitable to explore, from the designer viewpoint, a wide space of alternative solutions. One method, for a long time known, to describe the space organization in a building is the adjacency graph (March and Steadman [6]; Steadman [14]). By means of a planar graph it is possible to describe the adjacencies between different spaces in a building floor by representing the rooms with the vertices of a graph and the required adjacencies with the edges. This representation holds the relationships between the parts but does not hold information about the morphology of the design.

The "dimensionless" representation, firstly suggested by Eastman [3], allows to represent in a flexible manner the shape of a building plan or section. A dimensionless representation allows to define the topology of a geometric object without to fix the sizes of its parts. This kind of representation can be joined to an adjacency graph and viceversa, since it can be seen as an orthogonal embedding of the dual graph (the so called "plan graph") of the adjacency graph. Such an embedding is often optimal from some geometric viewpoint: e.g. it may contains a minimal number of internal angles (Tamassia [15]). A dimensionless representation allows to describe a whole class of objects that satisfy a particular set of geometric constrains.

Using variational geometry (Light and Gossard [5]) it is possible to generate several different geometric models starting from a given dimensionless embedding of topology. With such an approach it is necessary to set the geometric constraints between the parts of the model. This technique allows to manipulate the shape by calibrating some geometric values and by solving for the unbound variables. Constraints and degrees of freedom are fully contained in the description of the embedded topology of the object. A variational approach is an effective aid to morphology management because it allows to explore the consequences of the modification of an object part in comparison with the remainder.

Actually a true variational geometry based on constraint solving is quite hard to obtain in 3D, since the most significant design alternatives may completely change the topology, and hence the set of simultaneous constraints. A programming approach to geometry management, where the set of constraints do not necessarily rely on the topology, is instead possible.

3. Beyond variational geometry

The language PLaSM, which stands for PROGRAMMING LANGUAGE FOR SYMBOLIC MODELING, is a design language [9,10] which can be considered a geometric extension of a subset of the functional language FL developed by Backus and Williams and by their Functional Programming Group at IBM Research Division at Almaden (CA) [1]. PLaSM takes a programming approach to variational geometry and allows the designer to use any geometric expression in the language, i.e. any geometric shape, as the actual value for a parameter of any language function.

Multidimensionality Usually a solid modeler is only able to manipulate solids objects in 3D space. Actually, as we show later on, it is possible to integrate geometric modeling in 3D with geometric modeling in lower dimensions. In architectural design we are interested to work with 3D polyhedra as well as with 1D segments and with 2D polygons and polyhedral complexes.

PLaSM is dimension-independent [7,9], in the sense that it uses both a geometry representation and algorithms which can be applied to geometric objects of any intrinsic

dimensions, that is to points, curves, surfaces, solids as well as to manifolds of higher dimensions. With a language like PLaSM it is possible to define building plans, elevations and sections as 2D geometric models, often starting from 1D lateral dimensioning of 2D drawings, as well to automatically generate 3D models. Also, it is possible to assembly models of simple building parts to generate hierarchical assemblies of any complexity. In recent years multidimensionality is being considered a very important design goal for an advanced geometric modeler. Currently, PLaSM is one of few prototype geometric systems which actually implement multidimensionality.

Assembly functions Hierarchical models can be obtained in PLaSM by hierarchically assembling part models. When using the language, more detailed models can be obtained by part substitution and refinement starting from some template program.

The desired geometric models, generated by evaluation of a PLaSM program, are represented as HPCs (Hierarchical Polyhedral Complexes, see [11]), data structures which hierarchically collect a set of elementary polyhedra. Each part of such models is defined in local modeling coordinates and is mapped to the coordinates of the parent part at traversal time. Any model in the HPC representation is maintained as a *directed* and *acyclic multigraph*. An affine transformation is associated to each arc of such a graph. Every node is the root of a subgraph corresponding to a model part given in local modeling coordinates. Elementary polyhedra are associated to the leaves.

Partially open cell-complexes An elementary *polyhedron* is represented as a set of quasi-disjoint convex cells. A *polyhedral instance* is either the embedding and the affine map of a polyhedron or the embedding and the affine map of a polyhedral complex. A *polyhedral complex* is a set of quasi-disjoint polyhedral instances.

Each convex cell is represented as a set of facet equations. Facets of a *d*-dimensional cell are defined as *(d-1)*-dimensional affine subsets of its boundary. Each facet is uniquely associated with a facet *covector*. Each vertex is implicitly represented by the set of its incident facets.

In the HPC representation scheme it is easy to achieve cell decompositions where cells may have partially incomplete boundaries. This may be very useful in a geometric system[12]. For example, as it will be shown in the paper, it is so possible to automate the generation of building models with openings (doors and windows) starting from plans and sections, and by extracting the 2D skeleton (2D boundary facets of 3D cells) of some expression involving both plans and sections. If some 2D cells have *incomplete* boundaries (e.g. which define doors and windows in the layout) the openings in the 3D model will be automatically generated.

Generalized products PLaSM introduces a sort of algebraic calculus over geometric shapes by using algebraic topology (cell complexes) and some operations which are quite unusual in other geometric modeling systems. In particular, we have introduced a so called *generalized product* of polyhedral complexes [2], which allows as particular cases the cartesian product, the intersection, the extrusion, and the intersection of extrusions of polyhedral complexes of whatever dimension.

For example, the 3D model of a multi-floor building can be obtained by the cartesian product of its layout (2D complex) times a 1D complex which correspond to the lateral dimensioning of the floor widths and the inter-floor distances [10]. This computation can be expressed in PLaSM both as an algebraic expression, and as a language function which accepts two formal parameters corresponding to the layout value and to the lateral dimensioning value. In this second case, i.e. by extensively using functional abstractions, the design language can be semantically enriched and extended with respect to the application domain. The user only needs to know what is the meaning of the functions s/he is using and what parameters they expect.

4. Some design tools

In this section we quickly introduce some geometric programming tools which will be used in the example developed in the next section. For the language syntax and semantics the interested reader is referred to [9], where a glossary of primitive functions is also reported.

4.1 Dimensioning and positioning constraints

The dimensioning and relative positioning of design parts is done by using geometric *constraints*. In particular PLaSM allows for the use of design *variables* and constraint *operators*. Constraint operators are not primitive in the language, but can be defined by the user or included with predefined packages.

Part dimensioning by design variables The generation of the geometric shape of design parts can be done by instantiating functions and other language expressions with design variables. A design variable is a *0*-ary function (i.e. a constant) where a binding between a name and a value of a certain type is established. In PLaSM a value can be not only numeric, but of any other type, including geometric shapes and higher level functions. The use of the term variable is quite improper, since the language is purely functional, so that the binding between a name and the associated value cannot be changed at run time. In order to change such an association it is necessary to redefine the design variable, i.e. to give the language interpreter a new definition for it. Since the language interpreter maintains a network of functional dependencies between design parts, the redefinition of a design variable will imply a recomputation of all the design parts whose shape is dependent on that variable.

Positioning of design parts by constraint operators In PLaSM any design part is defined using local modeling coordinates. In order to give a hierarchical design component, i.e. a design component which is defined as an assembly of other design parts and subassemblies, the standard graphics method of hierarchical structures is used, with a semantics similar to that of PHIGS graphics system [4]. In particular, any assembly is defined as a sequence of geometric models and affine transformations, which must be applied to any models which follow in the sequence at traversal time. Each shape component in a structure network is so transformed within the coordinate system of the network root. This mechanism can be hidden to the user, which may use only binary constraint operators, so strongly increasing the readability of the symbolic description of a complex geometric shape.

User interface Operators can be written in infix form and composed hierarchically by mean of parentheses. E.g. it is possible to write expressions like

```
((shapeA op shapeB) op (shapeC op shapeD)) op shapeE
```

where the op symbol stands for any binary constraint operator. In particular, op can be any infix expression which takes values in a set of functions to be applied to the surrounding geometric arguments. In this paper we use the point matching operator

```
shapeA (pointOp ^ pointOp) shapeB
```

in order to impose the coincidence of the point returned by pointOp:shapeA with the point returned by pointOp:shapeB. In particular, pointOp is any function with input a geometric shape and output a point. So, the previous expression will generate the compound shape where shapeA and shapeB will match on the two specified points. In the following examples we have

$$\texttt{pointOp} \in \{\texttt{NE,SE,SW,NW,CE,CN,CS,CW}\ \}$$

where such functions return a geographic point with respect to the containment box of the input shape. Notice that `pointOp` can even return a fixed point. For example,

```
shapeA (K:<1.0,2.5> ^ NE) shapeB
```

is a constraint which forces the point at north-east of `shapeB` to match the point of coordinates `<1.0,2.5>` in the local coordinate frame of `shapeA`.

4.2 Polygons with incomplete boundaries

One input operation in the domain of CAAD modeling is described here. This allows the user to specify polygonal shapes with incomplete boundaries. So, the 2D model of a room can be specified as a convex polygon where each boundary side is fragmented into a 1D cell complex where some cells are empty. This approach will be used to model the openings in a room (see Section 5.3). Such a partially opened polygon has a HPC representation as a 2D cell decomposition with a *1*-cycle of triangular cells having a common vertex. Any pair of 1-adjacent cells in such a decomposition has no boundary on the common side. The side with no adjacency is defined either open or closed (with-no-boundary or with-boundary) depending on the user requirements.

User interface The user interface to the function which creates the HPC representation of such an object requires as input an alternating sequence with signed side dimensions and boundary angles. The dimensions are given positive when they correspond to a full cell of the boundary, and negative when they define an empty cell of the boundary. The angles are given in degrees and enclosed within a pair of angle brackets, in order to increase the readability of input data.

For example, the two polygons shown in Figure 1 are defined as follows:

```
Polygon: <12,<90>,6,<90>,5,-2,0.5,-2,2.5,<90>,2,-2,2, +>
Polygon: <9,<30>,3*3/SQRT:2,<60>,6,<90>,5,-2,0.5,-2,2.5,<90>,2,-2,2, +>
```

Notice that it is not necessary to give the last angle and side, which are automatically computed. The local coordinate frame is assumed with the *x* axis on the first polygon side, and with the origin on the first vertex of such a side. The angles are positive when counterclockwise.

Obviously, such polygons can be parameterized by using symbolic names for linear dimensions or angles. Also, the same symbolic name can be used for the common side of any pair of adjacent rooms, to be joined by using some constraint operator previously described. A whole layout plan, composed by both convex and unconvex rooms, can be so conveniently parameterized (see Section 5.3).

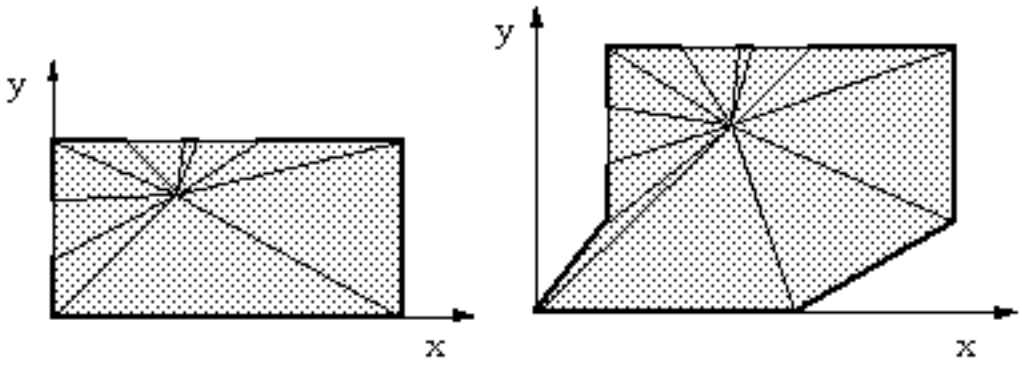

Figure 1: The polyhedral complexes generated by two applications of the `polygon` function to actual data.

4.3 Openings by intersections of complexes

The operation described in this section would not be necessary in a full implementation of the language, as described in [9]. In fact, in order to open the windows in the external envelope of the model or the doors in the internal partition, the Boolean difference of embedded polyhedral complexes would be sufficient. However, currently the Booleans are not fully implemented in PLaSM. A special operation is so required to create the openings for the windows in the envelope or the stairwell opening in the roofs.

User interface The user interface to such a design operation will be defined as a function application

```
window:<point,sides>:elevation
```

where `elevation` is any 2D complex and `point` is a pair of reals which gives the window displacement with respect to the containment box of the complex. Analogously, `sides` denotes the pair of lateral dimensions of the window. Also, according to the standard PLaSM syntax, an higher level function can be defined to open a set of windows:

```
DEF windows = COMP AA:window
```

So, e.g., a series of three windows in the 2D model of a building elevation can be given as

```
windows:<<p1,w1>,<p2,w2>,<p3,w3>>:elevation
```

4.4 Composition of multiple sections and plans

The more interesting features of the language concern its ability to automatically generate design descriptions of higher detail starting from lower detail ("partial") descriptions. In [10,2,9,8] it has been shown that 3D models of buildings can be automatically derived from their plans and sections by the so-called *intersection of extrusions* operation, which is a special case of the product operation described in [2].

Product of multiple plans and sections In order the generate the geometric model of a realistic building it is necessary to deal with more than one section. In particular we need to couple each building section and elevation with the appropriate portions of the plans of the building. This can be done by defining an appropriate partitioning of the plans with a family of open sets (i.e. sets without boundary) which can be thought as *open stripes* used to select the portions of plans to be operated with the various sections. Let $\boldsymbol{S}=\{S_i\}$, $\boldsymbol{P}=\{P_k\}$ be the sets of sections and plans, respectively. Let the set of open 2D stripes $\boldsymbol{O}=\{O_i\}$ be in a bijective correspondence with $\boldsymbol{S}$. Hence, the 3D model of the building described by $\boldsymbol{S}$ and $\boldsymbol{P}$ can be generated as an union of pairwise disjoint 3D cell complexes:

$$\cup_{ik}\, S_i \,\&\&\, (O_i \cap P_k)$$

each one generated by the intersection of extrusion (denoted by the binary infix operator &&) of a section S_i with an open plan stripe $(O_i \cap P_k)$ properly embedded in $\Re^3$. Actually, to automatically generate a 3D building model of some realistic complexity is much complex, as (a) there are changing 2D sections in both the x and y directions; (b) each section portion should be associated with a corresponding portion of a 2D plan and vice-versa; (c) 2D elevations should be taken into account, and associated to thin open stripes of plans.

Such characteristics of the problem can be considered by one-to-one associating each plan and section to a corresponding orthogonal open stripe of length 1 (to be considered as its *zone of influence,* and denoted here by the same index) and by associating each elevation to a thin orthogonal stripe of small length.

Then, consider the *ordered* sets $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$ obtained by ordering the elevations, sections and plans which are respectively orthogonal to the x,y,z axes. Let L^x, L^y, L^z be the associated open stripes. So, X_i and l_i^x will denote the elements in such sets. It is possible to see that the 3D model of the building, in the quite general case that the elevations and sections are orthogonal to the x and y axes, can be defined as the disjoint union of the cells

generated by a proper intersection of the extrusions. A sort of tensor product of plan, sections and open stripes can be given:

$$C_{ijk} = Y_j \;\&\&\; (((X_i \cap l_k^z) \;\&\&\; (Z_k \cap l_i^x)) \cap l_j^y)$$

So, the whole building model can be formally defined as a disjoint union $\cup_{i,j,k} C_{ijk}$ of open cell complexes.

User interface The user interface to the automated generation of 3D models is here defined by means of a function `volume` which requires as input a sequence of pairs $<<X, L^x>, <Y, L^y>, <Z, L^z>>$. Each input pair will contain both the ordered sequence of the sections perpendicular to a given coordinate axis, and the corresponding sequence of coordinates which delimit the open stripes associated to every section. So, if one such sequence contains *n* sections, the sequence of coordinates will contain *n+1* ordered values. For example, we have:

```
DEF model3D = volume: <<<z0_sec,z1_sec>,          <z0,z1,z2>>,
                        <<y0_sec,y1_sec,y2_sec>, <y0,y1,y2,y3>>,
                        <<x0_sec,x1_sec,x2_sec>, <x0,x1,x2,x3>> >
```

Notice that elevation layouts are defined exactly at the same way than internal floating sections. They are just associated to a smaller "zone of influence".

5. A case study: Wislocki house by Robert Venturi

The Trubek-Wislocki's house is a set of two houses designed by Robert Venturi in seventies (Sanmartin [13]). The Wislocki house is a building with a very simple layout plan. A central staircase organizes the volumes symmetrically, whereas the elevations are defined in opposition to the symmetry which characterizes the subdivision of internal spaces. For the present simulation of the design process we assumed constrained both the staircase position and the distribution of the various rooms.

5.1 Plan layout

Conversely to what assumed by default in the language, it will be useful to use symbolic instead than numeric identifiers for the three coordinate directions. So, we set:

```
DEF x = 1;  DEF y = 2;  DEF z = 3;
```

A set of reference data for the house is firstly defined. In particular we assign a reference value to the dimensions of the living room and of the balcony. The value of the balcony side will be assumed as the standard increment when experimenting some volumetric variations.

```
DEF X_living = 300;   DEF Y_living = 200;   DEF bal  = 50
```

As a consequence, the dimensions of the other spaces are given as functions of the reference elements. The stair is assumed squared, with side dimension `stair_side`.

```
DEF X_dining   = X_living;    DEF Y_dining = Y_living/2; DEF stair_side = Y_living/2
```

The stair is assumed as the "fixed" reference element for all the design alternatives. It is defined internal to other spaces (the living at the first floor, the passage at the second floor). So, a function which cuts the empty space for the stairwell within any other 2D polyhedral complex is defined. This function must also assign the same local coordinates to the corresponding empty spaces at the various levels. An appropriate translation is so applied to the argument of the function.

```
DEF stairwell (floor::ISPOL) = T:<x,y>:<stair_side - x_max, -:y_min>:floor & space
 WHERE  y_min = MIN:y:floor,  x_max = MAX:x:floor,  space = square_hole:
                  <-: X_living, X_living + delta, stair_side - (2 * delta), X_living,
                   -: Y_living, Y_living + delta, stair_side - (2 * delta), Y_living>
```

```
END
```

The layout plan of the first floor is so defined as “dining + living”, where the living contains the stairwell. The two spaces are coupled by imposing the matching on a point. In particular it is imposed that the “south-west point” of the living will coincide with the “north-west point” of the dining. The function `room_O` is used to define rectangular spaces with assigned lateral dimensions.

```
DEF dining      = room_O: <X_dining, Y_dining>             ;
DEF living       = (stairwell~room_O): <X_living, Y_living> ;
DEF first_floor = living (SW ^ NW) dining                  ;
```

Then the dimensions of the spaces of the second floor are given as functions of the reference dimensions of the first floor:

```
DEF X_bed          = X_living / 3;       DEF Y_bed          = (Y_living + Y_dining) / 2;
DEF Y_passage      = Y_living / 2;       DEF X_passage      = X_living - X_bed          ;
DEF X_bedC         = X_passage   ;       DEF Y_bedC         = Y_living - Y_passage      ;
DEF X_closet_bath = X_passage    ;      DEF Y_closet_bath = Y_dining                  ;
```

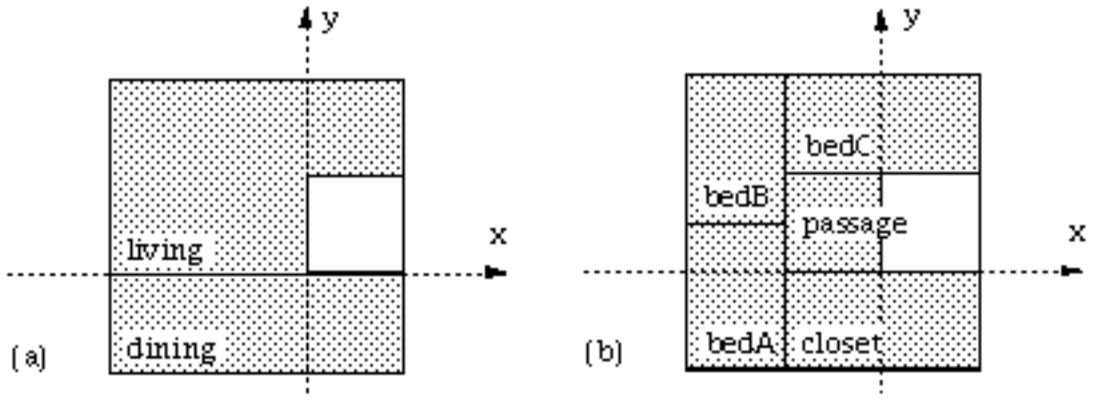

Figure 2: Layout plans of the first (a) and second (b) floors as generated by the initial definitions of `first_floor` and `second_floor`.

The spaces of the second floor are now defined. In this case the `passage` is the reference element which contains the stairwell:

```
DEF bedA         = room_O:<X_bed,Y_bed>                     ;
DEF bedB         = room_O:<X_bed,Y_bed>                     ;
DEF bedC         = room_O:<X_bedC,Y_bedC>                   ;
DEF passage      = (stairwell~room_O):<X_passage, Y_passage>;
DEF closet_bath = room_O: <X_closet_bath, Y_closet_bath>    ;
```

The layout plan of the second floor is assembled in two steps. First a `a_zone` and a `b_zone` are locally defined, then a definition of the second floor is given, where two reference points (called pivots) in both zones are constrained to match. In particular, the center point of the west side of the `passage` within the `a_zone` is matched with the point at north-east of the `bedA` within the `b_zone`.

```
DEF second_floor = a_zone (a_pivot ^ b_pivot) b_zone
 WHERE
   a_pivot = K:(CW: passage),  a_zone = passage (NW ^ SW) bedC (SW ^ NW) closet_bath,
   b_pivot = K:(NE: bedA)   ,  b_zone = bedA (NE ^ SE) bedB
 END
```

5.2 Volumetric studies

In order to study alternative shapes for the design, several 3D volumes are generated in this section. At this purpose, a 2D `section` of the whole house is defined by point matching of the sections of the first and second floor, respectively denoted as `sec_1` and `sec_2`. The first is simply defined as a translated rectangle; the second, which contains also the roof, is defined as a convex polygon with 5 vertices.

```
DEF Z_floor = 100;
DEF X_house = (2 * bal) + X_living;     DEF Y_house = (2 * bal) + Y_living + Y_dining;

DEF section =  sec_1 (NE ^ SE) sec_2;
DEF sec_1 = T:x:(-:(X_house/2 + stair_side/2)):(CUBOID:<X_house ,Z_floor>);
DEF sec_2 = T:<x,y>:<-:(X_house/2 + stair_side/2),Z_floor>:(MKPOL:<
                <<0,0>,                                          <X_house,0>,
                  <0,Z_floor>, <X_house/2, 2*Z_floor>, <X_house ,Z_floor>>,
                <1..5>, <<1>> >) ;
```

Volume solution A It is so possible to generate the first 3D volume solution by composing the 2D plans `first_floor` and `second_floor` with the 2D sections `sec_1` and `sec_2`, respectively. The `prod` operator is an alias for the intersection of extrusions after having embedded the 2D arguments into the coordinate subspaces z=0 and y=0 (see [2]). In particular, the second polyhedral argument of the `prod` operation is mapped in 3D so that its y (2D) coordinates coincide with the z coordinates of the 3D frame. So, the volumes of the first and second floor are separately generated; then they are collected within a structure.

```
DEF prod              = <x,y,0> && <x,0,y> ;
DEF vol_first_floor   = first_floor   prod   sec_1 ;
DEF vol_second_floor  = second_floor  prod   sec_2 ;
DEF house              = struct: <vol_second_floor, vol_first_floor> ;
```

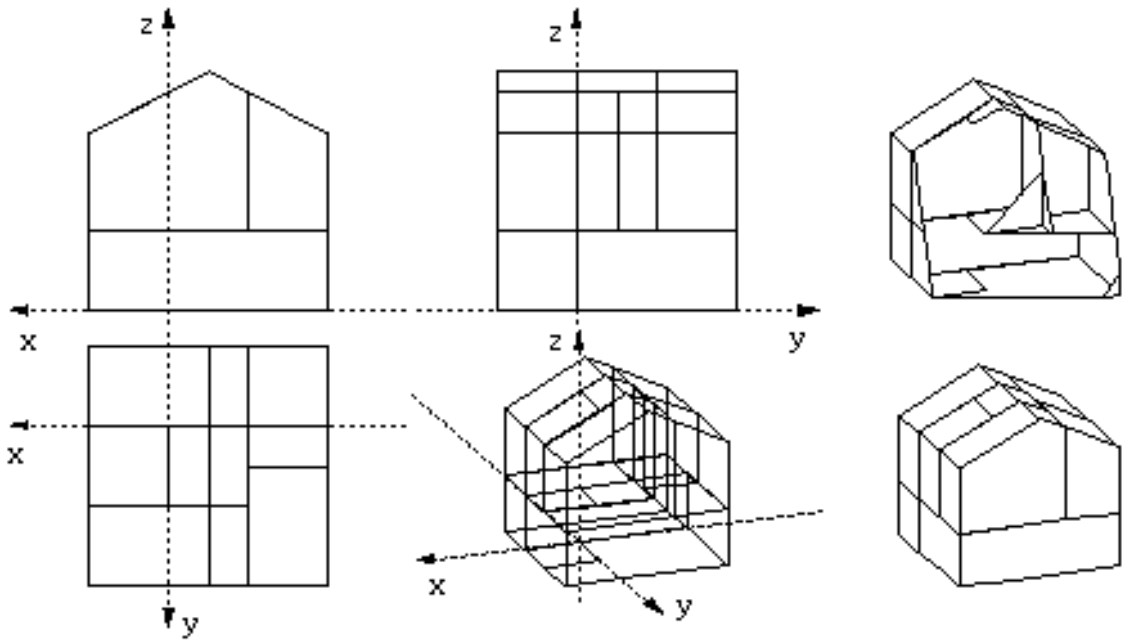

Figure 3: Model solution A generated by evaluating the expression `@2:house`.

Volume solution B Several volume alternatives are now given, always starting from the set I of definitions which have generated the solution A. For each new alternative, some rooms have a new definition. Then the corresponding 3D model is always generated by evaluating again the expression `@2:house`.

```
DEF closet_bath = room_O: <X_closet_bath + bal, Y_closet_bath + bal> ;
DEF bedC        = room_O: <X_bedC + bal, Y_bedC + bal>               ;
```

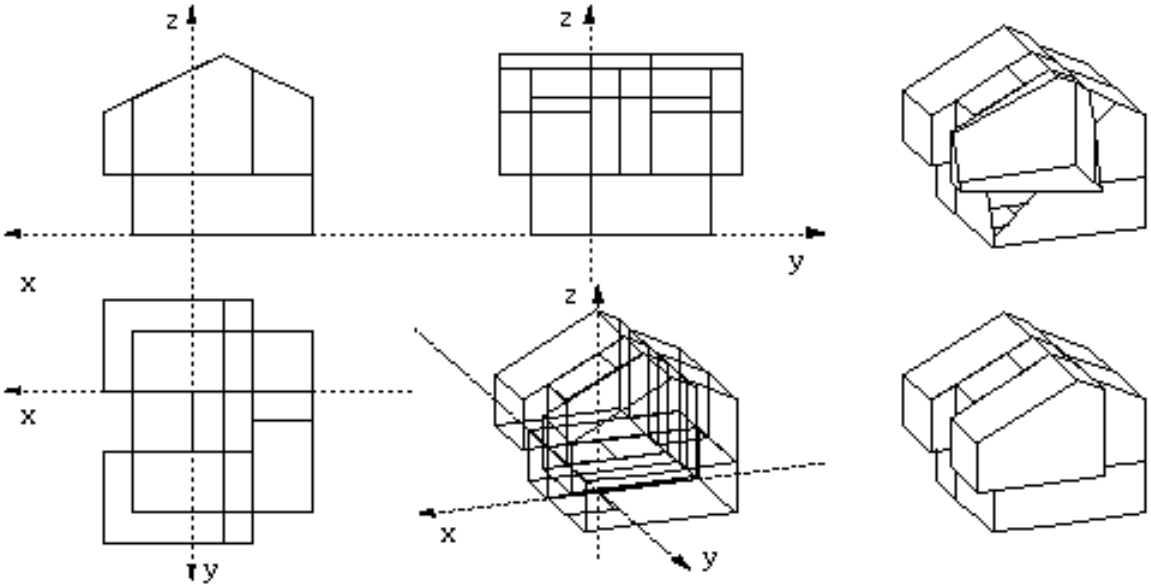

Figure 4: Model variation B generated by redefining the expressions for `closet_bath` and for `bedC`.

Volume solution C A third volume solution is also given by redefining some rooms. In particular, `bedA` and `bedB` are incremented on both sides of the balcony measure; the `living` is generated L-shaped, and the *x* direction of the `passage` at second floor is incremented of the half of the stair-side dimension. Then the 3D model is automatically generated.

```
DEF bedA    = room_O: <X_bed + bal, Y_bed + bal> ;
DEF bedB    = room_O: <X_bed + bal, Y_bed + bal> ;
DEF living  = (stairwell~room_L):
     <X_living - stair_side/2,  stair_side, stair_side, Y_living - stair_side>;
DEF passage = stairwell: (room_O: <X_passage + stair_side/2, Y_passage>);
```

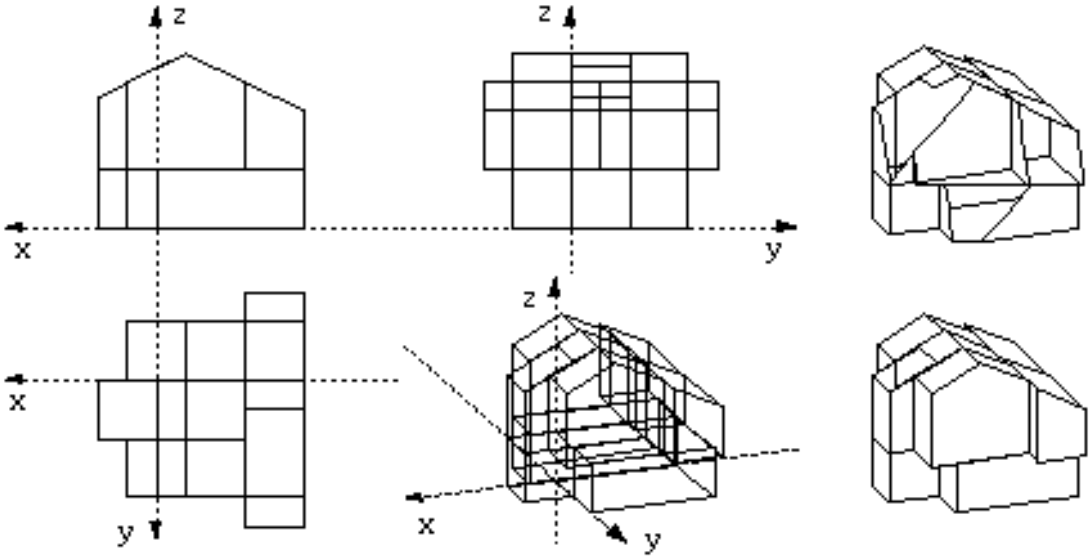

Figure 5: Model C generated by redefining the expressions for `bedA`, `bedB`, `living` and `passage`.

Volume solution D

```
DEF bedA = room_O: <X_bed   + bal, Y_bed   + bal> ;
DEF bedC = room_O: <X_bedC + bal, Y_bedC + bal>;
```

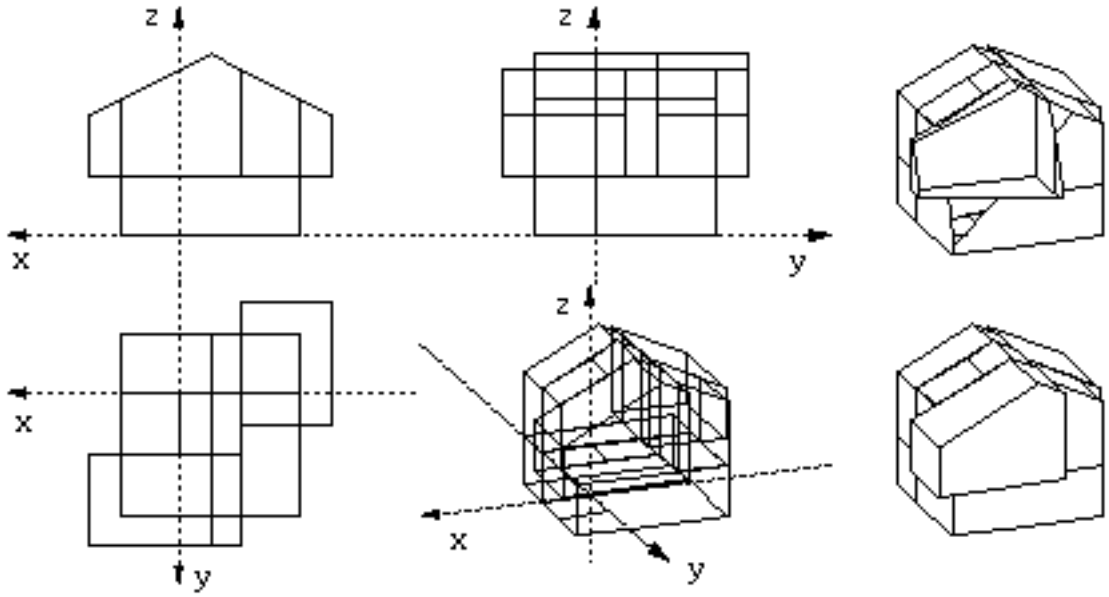

Figure 6: Model D generated by redefining the expressions for `bedA`, `bedB`.

Volume solution E In the solution E two bed rooms are enlarged on both sides, the `dining` is incremented in the *x* direction, and the living is rectangular, but is enlarged on both sides. The `stairwell` is opened inside the living.

```
DEF bedA  = room_O: <X_bed    + bal, Y_bed   + bal>;
DEF bedC  = room_O: <X_bedC  + bal, Y_bedC + bal>;
DEF dining = room_O: <X_dining + bal, Y_dining>;
DEF living = (stairwell~room_O): <X_living + bal, Y_living + bal>;
```

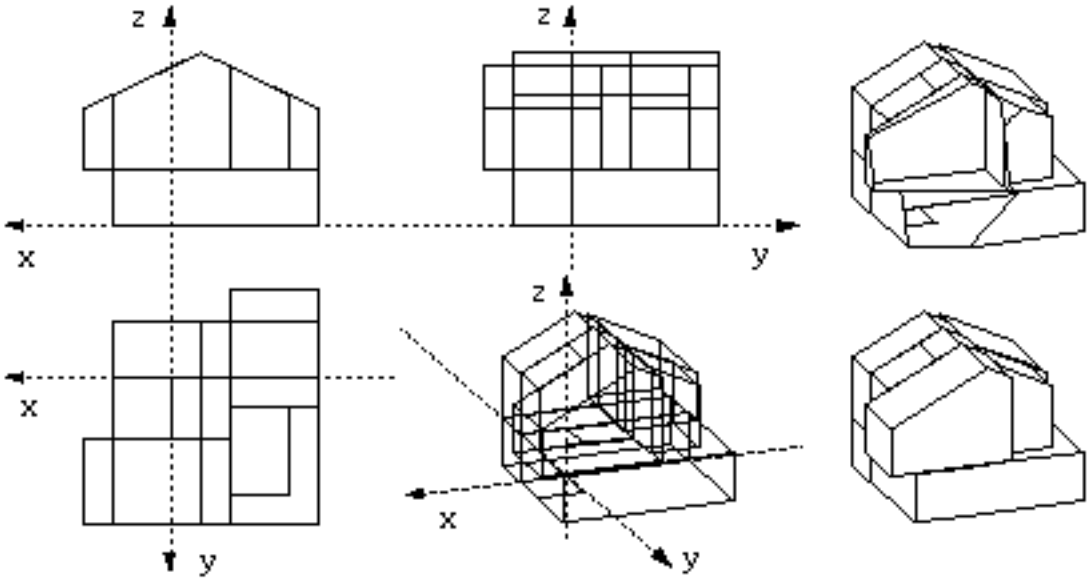

Figure 7: Model E generated by redefining the expressions for `bedA`, `bedC`, `dining` and `living`.

5.3 Openings in the internal partitions

In this section we demonstrate on a concrete example a method for opening passages between adjacent spaces in the 3D model automatically generated by product of plans and sections.

First, the "adjacency graph" of the plan is given, by listing in circular order the adjacencies of each room with the other rooms and the outside space. This representation could also be automatically derived from the 2D layout. Such definitions are given here for the second floor of the layout A (Figure 3). Notice that for each room the alternating sequence of linear partitions and angles must be given.

Each partition is bound to a design variable whose name contains the identifiers of the two spaces in contact along that partition. The sequence of partitions and angles is given counterclockwise, starting from either (a) the lower left partition or (b) a suitable initial angle which reduces the first partition to such a situation.

The `AdjGraph` function is simply obtained by composing the `CAT` primitive function with the `polygon` function. Notice that each space unit is defined in a local frame with the *x* axis along the first polygon edge and the origin at the lower left polygon vertex.

```
DEF AdjGraph = polygon~CAT                                                    ;
DEF bedA = AdjGraph:<bedA_out, <<90>>, bedA_closet, bedA_pass,<<90>>, bedA_bedB,<+>>  ;
DEF bedB = AdjGraph:<bedB_bedA, <<90>>, bedB_pass, bedB_bedC,<<90>>, bedB_out,<+>>    ;
DEF bedC = AdjGraph:< <<180>>, bedC_out,<<90>>, bedC_bedB,<<90>>, bedC_pass,<+>>      ;
DEF pass = (stair_space~AdjGraph):<
              <<180>>, pass_bedC ,<<90>>, pass_bedB, pass_bedA,<<90>>, pass_closet,<+>>;
DEF closet = AdjGraph:<<<180>>,closet_pass,<<90>>, closet_bedA,<<90>>, closet_out,<+>>;
```

Finally, the content of the design variable associated to each partition is specified, as a sequence of positive and/or negative numbers. Remember that positive numbers denote solid boundary elements, whereas negative numbers denote empty 1D cells in the boundary of the polygon.

```
DEF pass_bedB   = <-:Y_passage/2>  ;     DEF pass_bedA   = <-:Y_passage/2>          ;
DEF pass_bedC    = <stair_side,-:(X_passage-stair_side)/2,(X_passage-stair_side)/2> ;
DEF bedA_closet = <Y_closet_bat/2, -:(Y_closet_bat/2)>                              ;
DEF bedA_out    = <X_bed>          ;     DEF bedA_bedB   = <X_bed>                  ;
DEF bedB_bedC   = <Y_bedC>         ;     DEF bedB_out    = <X_bed >                 ;
DEF bedC_out    = <X_bedC>         ;     DEF closet_out  = <X_passage>              ;
```

Since partitions between pairs of internal spaces (i.e. different from the outside space) must be given twice, with the two opposite orientations, the design variables `pass_closet`, `bedA_pass`, `closet_pass`, `closet_bedA`, `bedB_bedA`, `bedB_pass`, `bedC_bedB` and `bedC_pass` are computed respectively as `REVERSE`, of the design variables `pass_bedC`, `pass_bedA`, `pass_closet`, `bedA_closet`, `bedA_bedB`, `pass_bedB`, `bedB_bedC` and `pass_bedC`. E.g. `DEF bedB_bedA = REVERSE: bedA_bedB.`

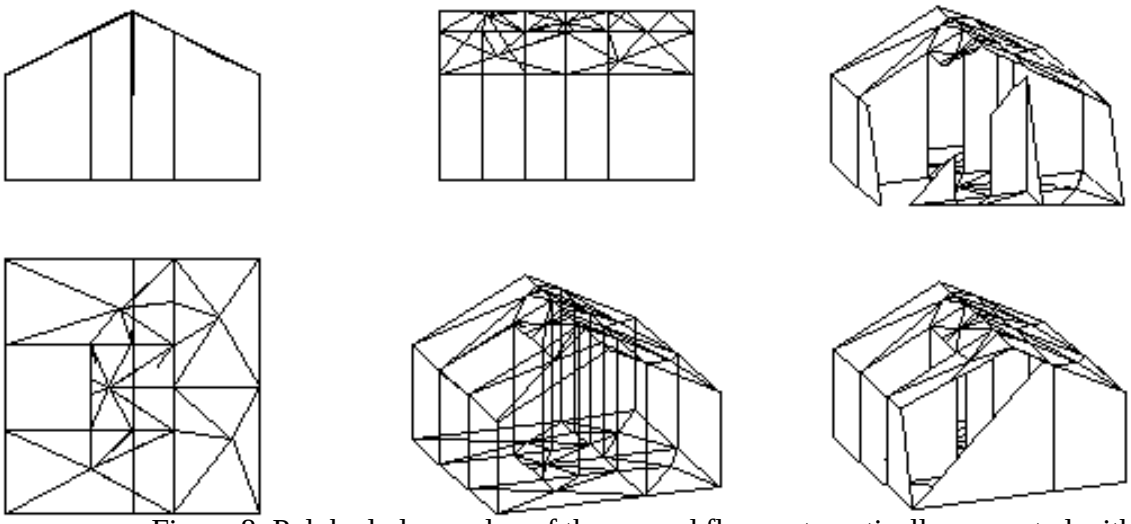

Figure 8: Polyhedral complex of the second floor automatically generated with the user-defined openings on the internal partitions.

5.4 Elevations and sectioning

In this section we define some building elevations and sections as 2D polyhedral complexes. In the next section they will be used to automatically generate a more detailed 3D model of the building, i.e. a model containing also the openings (windows and doors) within the building envelope.

So, a set of real dimensions for the main geometric element of the house is now given. The whole house model is parameterized on such dimensions. If some dimensions are changed, the only elements which either directly or indirectly depend on them are computed again.

The first floor is defined at *z* height 0; the second floor is at height `z_floor1`; the bottom of roof is at height `z_floor2`; the top at height `z_floor3.` True dimensions are now given for the living room, and consequently for the other rooms of the house.

```
DEF X_living = 620; DEF Y_living = 740 * 2/3; DEF bal      = 100;
DEF z_floor0 = 0  ; DEF z_floor1 = 265       ; DEF z_floor2 = 180; DEF z_floor3 = 300;
```

Another set of definitions is given for the possible sizes of doors and windows (`d1,...,d5`), as well for the dimensioning of the front and lateral elevations.

```
DEF d1 = 15 ; DEF l1 = bal+d1           ; DEF h0 = -:delta        ;DEF ll2=bal+2*d1+d4;
DEF d2 = 110; DEF l2 = bal+2*d1+d2      ; DEF h1 = d4 - d2        ;
DEF d3 = 90 ; DEF l3 = bal+3*d1+2*d2    ; DEF h2 = z_floor1 + d2  ;
DEF d4 = 220; DEF l4 = bal+4*d1+2*d2+d3; DEF la = bal+x_bed-d1-d2;
DEF d5 = 150; DEF l5 = bal+5*d1+3*d2+d3; DEF lb = bal+x_bed+d1    ;
```

The floating front section, named `front_sec`, perpendicular to the *y* axis in the 3D volume (see Figure 9 and Figure 10), is defined as a 2D primitive polyhedron with a single convex cell. In particular it is given as the translated convex hull of five extreme 2D points:

```
DEF front_sec = T:x:(-:(x_house - stair_side - bal)):(MKPOL:<
                    <<0, 0>, <x_house, 0>, <0, z_floor1 + z_floor2>,
                     <x_house/2, z_floor1 + z_floor2 + z_floor3>,
                     <x_house, z_floor1 + z_floor2>>,
                   <1..5>,<<1>>>) ;
```

The first front section `y0_sec` is defined by opening a series of windows in `front_sec`. The front section `y0_sec` will be valid in the *y* interval between `y0` and `y1`.

```
DEF y0_sec = windows:<<<l1,h1>,<d2,d2>>,  <<l2,h1>,<d2,d2>>,  <<l3,-:delta>,<d3,d4>>,
                     <<l4,h1>,<d2,d2>>,  <<l5,h1>,<d2,d2>>,
                     <<la,h2>,<d2,d5>>,  <<lb,h2>,<d2,d5>> >: front_sec;
```

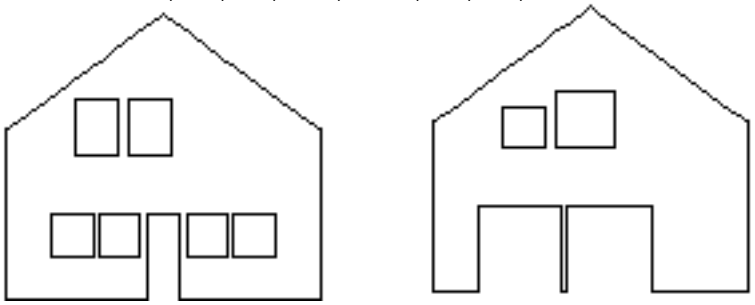

Figure 9: Front elevations `y0_sec` and `y2_sec`.

An alias `y1_sec` for the front section will be valid between the coordinates `y1` and `y2`:

```
DEF y1_sec = front_sec;
```

A last `y2_sec` front section, corresponding to the other extreme front elevation, will be valid between the extreme values `y2` and `y3`.

```
DEF y2_sec = windows: <<<l1,h0>, <d4,d4>>, <<ll2,h0>,<d4,d4>>,
                      <<la,h2>, <d2,d2>>, <<lb ,h2>,<d5,d5>>>: front_sec;
```

The floating side section, which is perpendicular to the *x* axis of the 3D frame, is defined (in 2D) as a translated rectangle, of size `Y_house` × `(z_floor1 + z_floor2 + z_floor3)`.

```
DEF s_sec = t:x:(-:((Y_house  - stair_side) / 2))):
                (CUBOID:< Y_house ,z_floor1 + z_floor2 + z_floor3>);

DEF x0_sec = windows:<<<lc,h2>,<d2,d2>>,  <<ld,h2>,<d2,d2>>>:s_sec
  WHERE
    lc = bal + y_bed - d1 - d2 ,       ld = bal + y_bed + d1
  END;

DEF x1_sec = s_sec          ;   DEF lle = bal + y_bed - d1 - d2 ;
DEF he = z_floor1 - (d2/2) ;   DEF lf  = bal + y_bed + d1      ;
DEF hf = z_floor1 +  d1    ;

DEF x2_sec = windows:<<<l1,h2>,<d2,d2>>,  <<lle,he>,<d2,d2>>,  <<lf,hf>,<d2,d2>>>:s_sec;
```

5.5 Generation of the 3D Volume

Finally the 3D models for the house solution A and E are generated. With reference to the methodology described in Section 4.4 the function `volume` is used at this purpose.

```
DEF model3D = volume :< <<first_floor, second_floor         >, <z0, z1, z2    >>,
                         <<y0_sec     , y1_sec      , y2_sec>, <y0, y1, y2, y3>>,
                         <<x0_sec     , x1_sec      , x2_sec>, <x0, x1, x2, x3>> >
  WHERE
    z0 =-:delta,        z1 = z_floor1    ,  z2 = z_floor1+z_floor2+z_floor3 + delta ,

    y0 = -:(Y_dining + delta)           ,  y1 = y0 + 2* delta                      ,
    y2 = y_living - delta               ,  y3 = y2 + 2* delta                      ,

    x0 = -:(x_living-stair_side)-delta ,  x1 =  x0 + 2 * delta                     ,
    x2 =  stair_side - delta            ,  x3 =  x2 + 2 * delta
  END;
```

Figure 10: 3D embedding of plans and elevations which produce the A volume solution.

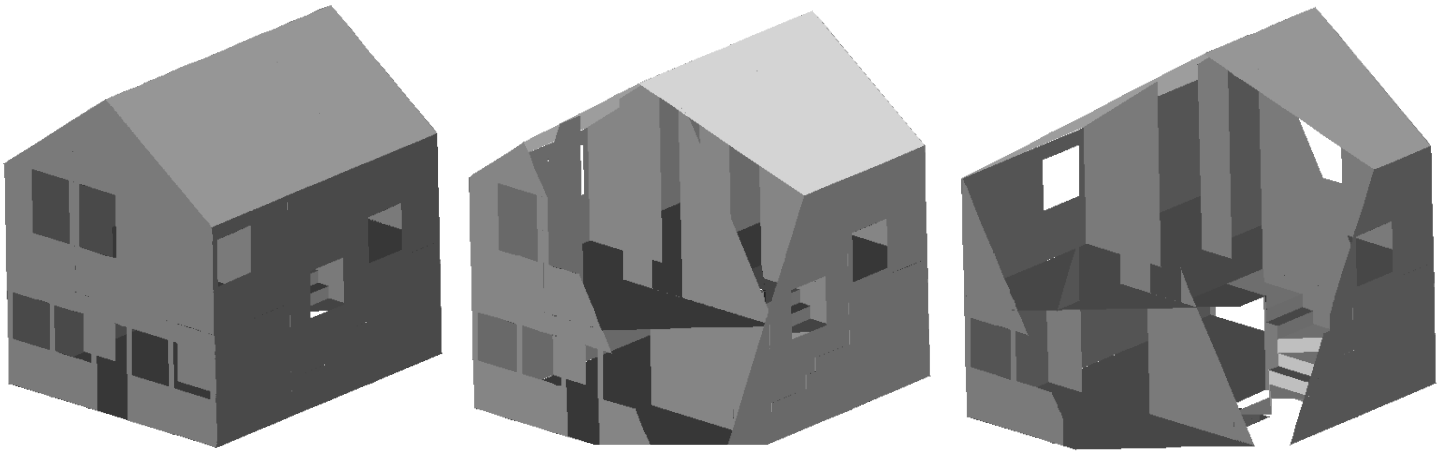

Figure 11: 3D volume solution A.

The evaluation of the `model3D` function within the functional environment associated to the A solution will result in the model displayed in Figure 11.

To generate the 3D model E the elevations displayed in Figure 12 are needed.

By evaluating the `model3D` function in the functional environment of the E solution will result in the model of Figure 13.

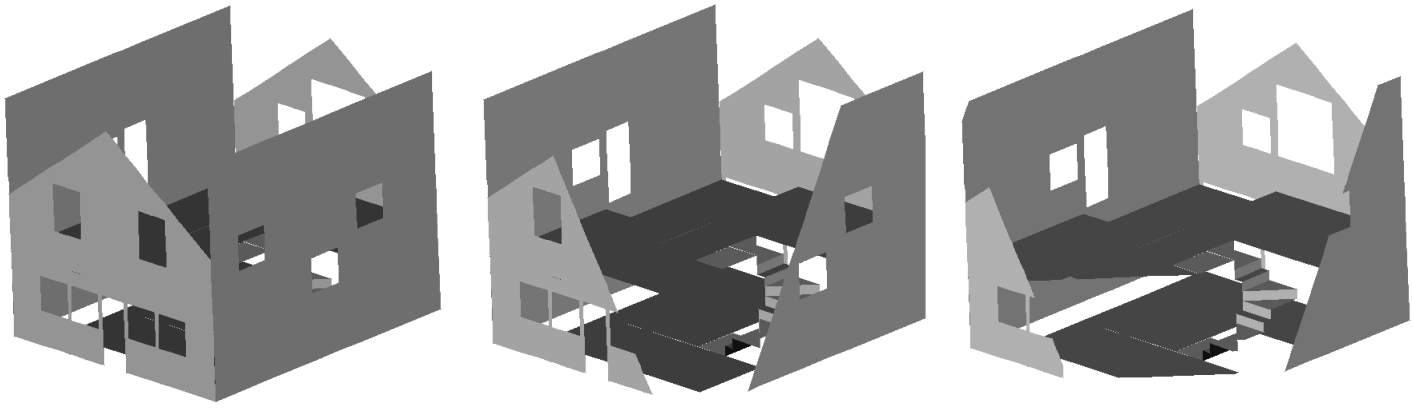

Figure 12: 3D embedding of plans and elevations which produce the E volume solution.

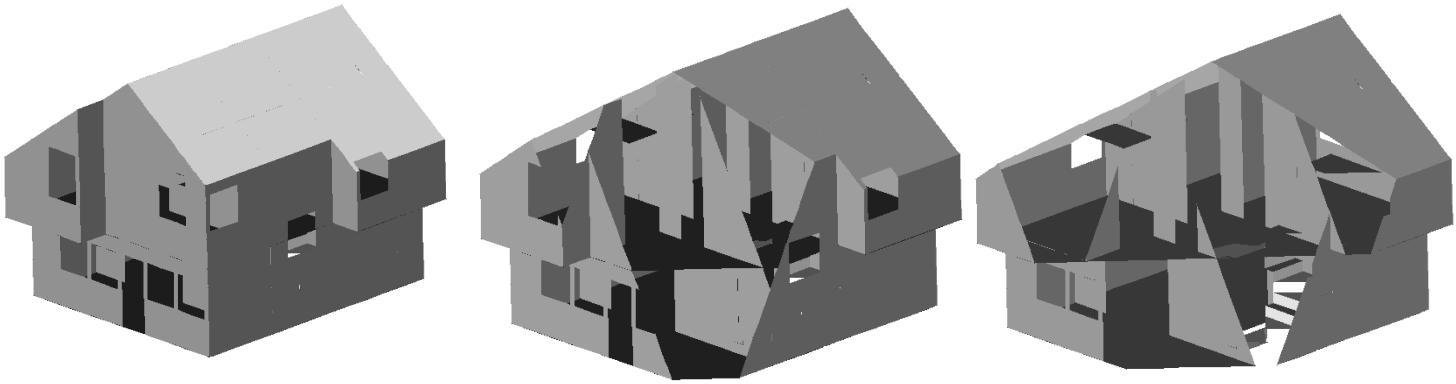

Figure 13: 3D volume solution E.

References

[1] BACKUS, J., WILLIAMS, J.H. AND WIMMERS, E.L. An Introduction to the Programming Language FL. In *Research Topics in Functional Programming*, D.A. Turner (Ed.), Addison-Wesley, Reading, MA, 1990.

[2] BERNARDINI, F., FERRUCCI, V., PAOLUZZI, A., AND PASCUCCI, V. Product operator on cell complexes. In *Solid Modeling '93, Second ACM/IEEE Symposium on Solid Modeling and Applications* (Montreal, CA, 1993), J. Turner, J. Rossignac, and G. Allen, Eds., ACM Press, pp. 43--52.

[3] EASTMAN, C.M. Representation for space planning. *Communications of th ACM 13*, (1970), 242--1970.

[4] HOWARD, T., HEWITT, W., HUBBOLD, R., AND WYRWAS, K. K. *A Practical Introduction to PHIGS and PHIGS PLUS.* Addison Wesley, Reading, MA, 1991.

[5] LIGHT, R. AND GOSSARD, D. Modification of Geometric Models though Variational Geometry. *Computer Aided Design 14*, 4 (1982), 209--214.

[6] MARCH, L. J., AND STEADMAN, J. P. *Architectural Morphology*. RIBA Publications, London, 1971.

[7] PAOLUZZI, A., BERNARDINI, F., CATTANI, C. AND FERRUCCI, V. Dimension-Independent Modeling with Simplicial Complexes. *ACM Transactions on Graphics 12*, 1 (1993), 56-102.

[8] PAOLUZZI, A., AND PASCUCCI, V. Building Design Programming with a Functional Language. *6th Conference on Computing in Civil and Building Engineering* (Bauforum, Berlin, 1995). To appear.

[9] PAOLUZZI, A., PASCUCCI, V., AND VICENTINO, M. Geometric programming: A programming approach to geometric design. *ACM Transactions on Graphics*, accepted for publication, 1995.

[10] PAOLUZZI, A., AND SANSONI, C. Programming language for solid variational geometry. *Computer Aided Design 24*, 7 (1992), 349--366.

[11] PASCUCCI, V., FERRUCCI, V., AND PAOLUZZI, A. Dimension-independent convex-cell based HPC: Representation scheme and implementation issues. In *Solid Modeling '95, Third ACM/IEEE Symposium on Solid Modeling and Applications* (Salt Lake City, Utah, 1995), C. Hoffmann and J. Rossignac, Eds., ACM Press, pp. 163--174.

[12] ROSSIGNAC, J.R. AND O'CONNOR, M.A. SGC: A Dimension-independent model for pointsets with internal structures and incomplete boundaries. *Geometric Modeling for Product Engineering*, Proceedings of the 1988 IFIP/NSF Workshop on Geometric Modelling, Rensselaerville, NY, September 18-22, 1988. M. J. Wozny, J.U. Turner and K. Preiss (Eds.), North-Holland, pp.145--180, 1990.

[13] SANMARTIN, A. *Venturi, Rouch & Scott Brown.* Academy Editors, London, 1986.

[14] STEADMAN, J. P. *Architectural Morphology*. Pion, London, 1983.

[15] TAMASSIA, R. On Embedding a Graph in the Grid with the Minimum Number of Bend. *SIAM Journal on Computing 16*, 3 (1987), 421--444.