Building Design Programming with a Functional Language *

A. Paoluzzi
Dip. di Disc. Scientifiche: Sez. Informatica, Terza Università,Via Segre 3, 00146 Roma,, Italy

V. Pascucci
Dip. di Informatica e Sistemistica, Università "La Sapienza",Via Buonarroti 12, 00185 Roma,Italy

ABSTRACT: A quite complex example of building design programming is discussed in this paper. In particular we introduce some operators which allow to easily define parametric layout plans, where both single and grouped rooms are constrained both on their linear dimensions and on the relative positioning. Such operators are then used to generate constraint-based models of the hierarchical set of spaces which define a terraced building. The given PLASM description is completely coordinate free, i.e. all design parts are *intrinsically* defined. The relative positioning and dimensioning of parts is obtained using constraints. The whole set of design choices, i.e. the *design knowledge*, is compactly codified within the easily readable PLASM description of each plan and section of the building. The 3D model of the building is then *automatically* generated from plans and sections. The authors believe that such a natural and semantically rich description may be very useful in the design process, mainly if coupled with a very high level (intelligent) graphical interface (to be developed).

1 Introduction

The term "Geometric Programming" was introduced to denote the functional approach to geometric design programming with the language PLASM developed by the CAD Group at the University of Rome [7, 5]. This language was originally developed as a language for building design, but is being currently experimented also as a general purpose language for modeling highly complex geometric shapes. It can be roughly considered as a geometry-oriented extension of a subset of the functional language FL, developed by Backus, Williams and their group at IBM Almaden [2, 1].

In this paper we discuss the modeling of a building design as a PLASM *script*. This modeling was done starting from the original set of plans, sections and orthographic views. The modeled project (see Fig. 6) is a terraced housing realized in Bologna (Italy) by the Prof. Arch. E. Zambelli at the end of seventies. The terraced housing is composed by the aggregation of a certain number of two-floor houses with basement.

The main steps of our design modeling can be summarized as follows:

- plan analysis and definition (in 2D) for the single housing unit;
- section analysis and definition (in 2D) for the single housing unit;
- combination of plans and sections, by using the product operator described in [3], in order to automatically generate the volumes of the housing units;
- staircase definition (in 3D) to be matched with the stairwell of the housing units;
- aggregation of a set of house pairs, in order to generate the complete model of the terraced housing.

It is important to note that PLASM was developed not as a graphic language for modeling and simulation but as a true "design language", with the aim of supporting all kinds of designing activities, which consist in shape analysis and synthesis, as well as of supporting the design revision steps. In order to stress this aspect, we choose to discuss the design synthesis at a very high level, as the architect could do in the initial stages of design. This results in a variational reconstruction of a schematic model of the volumes of the housing project. In particular, we will develop our schematic housing model as a set of space polygons, automatically generated starting from 2D plans and sections, but without the opening of doors and windows (at least currently). Some other simplifications will be indicated at their place in the paper.

A more detailed model could be generated at the price of writing much more code, with a quantity of details which cannot be discussed in the present paper. Our aim in developing the PLASM language was also to allow as much as possible for automatic translation

*This work was partially supported from Italian Research Council, with contract n. 91.03226.64, within the "PF Edilizia" Project.

This needs some more research and development effort of the PLASM environment. In our current plans we suppose to reach this major goal in two or three years.

Notice also that we developed the PLASM model without any interaction with the designer, but only starting from the detailed plans and sections of the building. So, the PLASM code will reflect our reading of the design, which do not necessarily coincide with the designer view. In other terms, the design decomposition we give in the paper only reflects our reading of the design, and corresponds to only one of the several ways to build the geometric model of the schematic building. Anyway, a closer interaction with the designer would allow for programming his true view of design development process, starting from the initial ideas and "feeding back" everywhere this should be necessary in order to satisfy some design constraint.

As the following discussion should make clear, the language seems to have an amazing descriptive power, and should allow the designer to implement in few lines of (generative and variational) PLASM code both the structure of the design and the internal and external constraints acting on the geometric shape. Actually, the designer should not be required to be also a computer specialist, but the language should be embedded into a modern interactive user interface, where all the design intentions could be graphically captured and automatically coded as a PLASM script.

1.1 *Preliminary definitions*

Every housing unit in the project contains a basement, a first floor and a second floor. It is possible to see that the building is made by the aggregation of a fundamental housing unit (see Fig. 1), which is repeated in groups of two or four units at a time. Each group of two or four units is obtained by one or two specular reflection of the same basic unit. Actually, as we show in the following, two minor variations are given of the basic housing unit.

After a first analysis of the design layout we decided to write a few general PLASM operators to be extensively used to model the plans. Such functions of general utility must not be written at any new project, but can be stored into persistent files called PLASM packages (as they constitute a sort of general design knowledge) which are included and used in several different design projects. Such operators are here given for sake of completeness.

The space element which is more frequently instanced in the design is a rectangular space, which will be denoted as `O_space` and will be defined as an alias for the primitive PLASM function `CUBOID`:

```
DEF O_space = CUBOID
```

In order to define space units which have a more complex hierarchical shape, it is useful to have a function which can be used to specify the relative positioning of (hierarchical) aggregates of spaces. In particular a binary function "^" is given which allow to match two shapes on a specified point (see Fig. 2). Often, but not necessarily, these points will coincide with one of

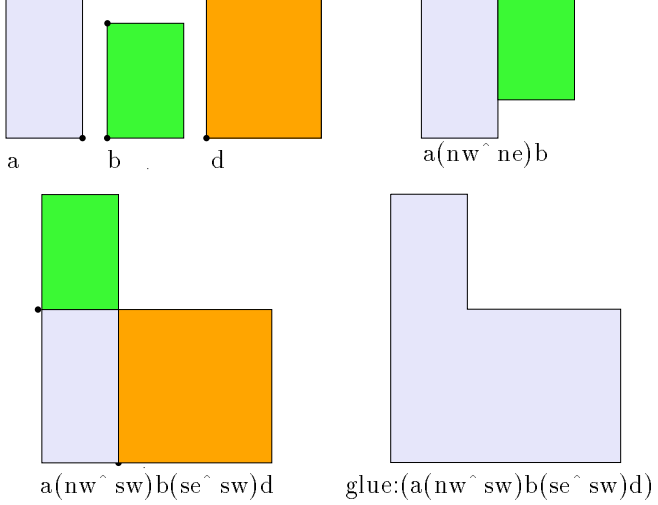


Figure 2: Matching and glue of rectangles.

the extreme "geographic" points of the containment boxes of the two shapes to be matched. E.g., let `sw`, `se`, `nw`, `ne` be four "geographic" functions which compute the lowest left, the lowest right, the highest left and the highest right point of a 2D shape ("polyhedral complex" — see [6]), respectively:

```
DEF sw = [MIN:1, MIN:2]; % south-west %
DEF se = [MAX:1, MIN:2]; % south-east %
DEF ne = [MAX:1, MAX:2]; % north-east %
DEF nw = [MIN:1, MAX:2]  % north-west %
```

The definition of the "^" operator will be given as follows, where an affine transformation matrix will be automatically included in a sequence containing the two shapes to be matched, according to the semantics of the ISO standard graphic system PHIGS [4]:

```
DEF ^ (geo1,geo2::IsFun) =
 STRUCT ~ [S1, (T:<1,2>) ~ parameters,S2]
 WHERE
  parameters = (AA:-)~TRANS~[geo1~S1,geo2~S2]
 END
```

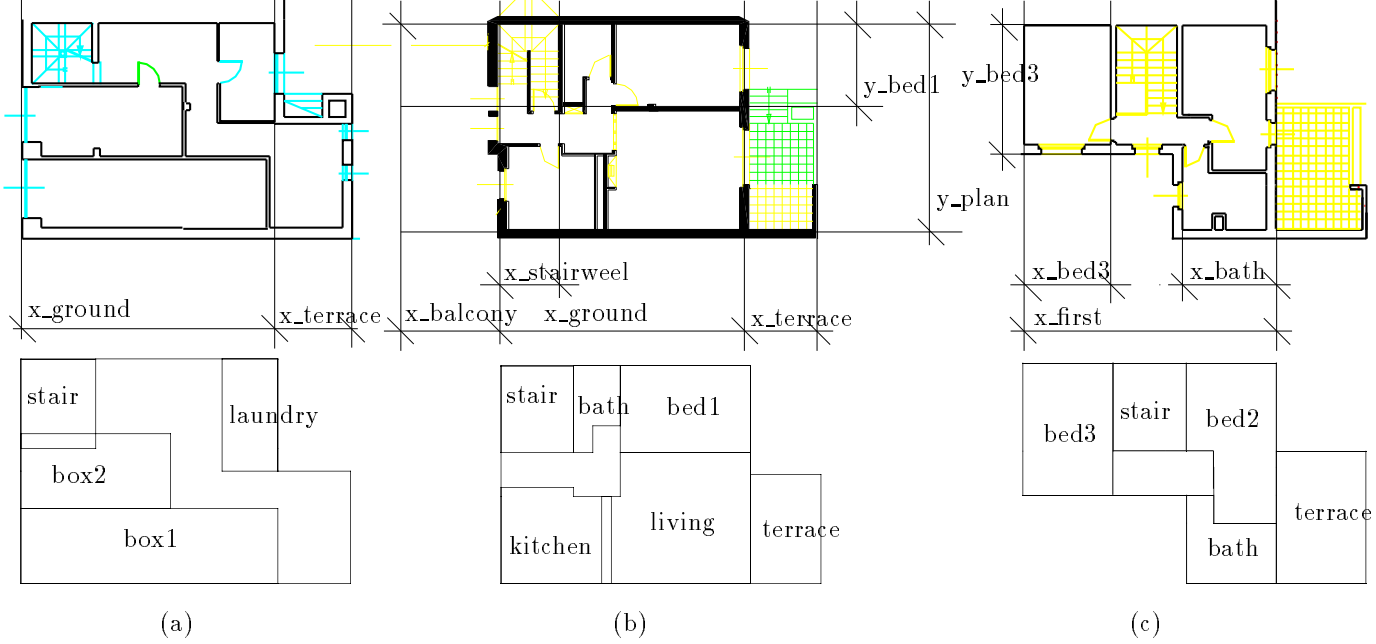
where the formal parameters `geo1` and `geo2` are usually geographic functions which compute the two points on the argument shapes which have to coincide in the resulting structure. The PLASM script which codifies the matching operation described in Figure 2 will be written as follows:

```
DEF a = O_space:<10,20>;
DEF b = O_space:<10,15>;
a (nw ^ ne) b
```

In order to group three spaces `a`, `b` and `d` in such a way that `b` is put over `a` and `d` to the bottom of `a`, the following PLASM script (see Fig. 2) is defined.

```
DEF d = O_space:<15,20>;
a (nw ^ sw) b
  (se ^ sw) d
```

To understand this example it is necessary to say that the standard order of execution of the operations within an expression (without parenthesis) is from left to right. Hence, the whole group of spaces is defined in the local coordinates of the first shape; then the second shape is placed in this system; then the third shape is matched with a geographic point computed on the



(a) (b) (c)

Figure 1: The three plans of type B. Basement (a), first floor (b) and second floor (c) and their corresponding polyhedral approximations.

group of the first two shapes, and so on. In order to glue the polyhedral cells of a polyhedral complex into a single element without internal partitions, a special-purpose `glue` function is defined:

```
DEF glue = MKPOL~[S1,S2,[FROMTO~[K:1,LEN~S2]]]
                ~UKPOL
```

This function just compute (using the primitive function `UKPOL` — see [6]) a symbolic description of the polyhedral complex which it is applied to, and then slightly modify it, in order to put together all the convex cells into a single polyhedral cell, and finally commits the reconstruction of the internal data structure to the predefined function `MKPOL`.

It is so possible to easily define a new parametric function `L_space` which compute the geometric model of a 2D L-shaped space starting from the lateral dimensions, according to the generative scheme of Figure 2, which we have just discussed.

```
DEF L_space (x1,x2,y1,y2::IsReal)=
 glue:(O_space:<x1,y1> (se ^ sw) O_space:<x2,y1>
                       (nw ^ sw) O_space:<x1,y2>)
```

The last 2D shape of Figure 2 can be so computed by evaluating the PLASM expression:

```
L_space:<10,15,20,15>
```

1.2 *Schematic layout plans*

First of all, we define a set of numeric parameters, to be used for common dimensioning reference of the design elements of a single housing unit (see Fig. 1):

```
DEF y_plan   = 802.5 % lenght of plans (y axis)%;
DEF x_ground = 920  %width of ground floor (x) %;
DEF x_terrace = 260 %  width of terrace        %;
DEF x_balcony = 350 % distance first floor -
                              center of balcony %;
DEF y_bed1 = y_plan * 2/5 % lenght of bedroom  %;
DEF x_bed1 = y_bed1 * 3/2 % bed aspect ratio   %;
DEF x_stairwell = y_plan / 3% stairwell lenght %
```

Then two global variables are defined which state the width of passage spaces (`passage`) and a "small" dimension (`delta`) with respect to the sizes of the design spaces:

```
DEF passage = 100 ;
DEF delta   =   1
```

The 3D staircase of the whole housing unit will be defined later on in the paper. At this point, i.e. while defining the plan layouts, it is sufficient to leave an empty space within the 2D schematic floors. So, a geometric function `floor_base` is defined which returns a 2D polyhedral complex, with an empty cell corresponding to the stairwell. This `floor_base` will be used to compute by intersection the true geometric shapes of the floors at the various levels in the housing unit.

```
DEF floor_base = (glue~STRUCT):
  <QUOTE:<x_balcony+delta,-:(x_stairwell-delta),
          x_ground> *
   QUOTE:<y_plan-y_bed1,y_bed1-delta,delta>,
   QUOTE:<-:(x_balcony+delta),x_stairwell-delta>*
   QUOTE:<y_plan-y_bed1,-:(y_bed1-delta),delta>>
```

Notice that the empty space corresponding to the staircase is defined smaller (of `delta`) than the actual dimensions of the staircase, in order to mark its perimeter. By using the design elements defined at this point, it is now possible to give the layout plan of the first floor:

```
DEF first_floor =
    floor (se ^ se) (bath (ne ^ nw) bed
                          (se ^ ne) living)
          (sw ^ sw) kitchen
          (se ^ sw) terrace
WHERE
 floor = floor_base & T:1:x_balcony:
                      (O_space:<x_ground,y_plan>),
 bath  = S:2:-1:(L_space:
           <x_ground-x_bed1-x_stairwell-passage,
               passage,y_bed1-passage,passage>),
```

```
 living  = S:1:-1:(L_space:<x_bed1,passage*2/3,
                      y_bed1,y_plan-(2*y_bed1)>),
 kitchen = L_space:<x_stairwell,
         x_ground-x_stairwell-x_bed1-(passage/3),
                              y_bed1,passage/3>,
 terrace = O_space:<x_terrace,y_plan /2>
END
```

The plan of the first floor is made by four spaces: `bed`, `living`, `bath` and `kitchen`. The first only is defined as a `O_space`, whereas the others are defined as `L_space`. The `bath` is obtained by reflection of the prototype shape with respect to the first axis (`s:2:-1`). Analogously, the `living` is obtained by reflection of the prototype shape with respect to the second axis (`s:1:-1`). Notice that (`s:1:-1`) is the scaling on the first coordinate for the scaling parameter -1, which results in the appropriate reflection. The geometric model corresponding to the generated shape (see Fig.1) can be obtained by evaluating the expression:

```
first_floor
```

At the same way can be defined the layout plan of the basement of the housing unit (see Fig. 1).

```
DEF basement_floor =floor(sw ^ sw)internal_spaces
WHERE
 floor = floor_base &
              T:1:x_balcony:(L_space:<x_ground,
                  x_terrace,y_plan/2,y_plan/2>),
 internal_spaces = box1 (nw ^ sw) box2
       (ne ^ k:<y_plan/4,x_stairwell/2>) laundry,
  box1    = O_space:<x_ground,x_stairwell> ,
  box2    = O_space:<y_plan * 2/3 ,x_stairwell >,
  laundry = O_space:<y_plan/4,y_plan/2 >
END
```

In the last definition the dimensions of the basement have been constrained to those of the first floor, by making use of the same variables. In particular, it has been stated that the two plans have the same length (`y_plan`), whereas the basement width will be obtained by adding `x_terrace` to that of the first floor, i.e. it is defined as `L_space:<x_ground, x_terrace, ...>`. The two basement boxes and the stairwell have the same length, as it was stated that `x_stairwell` = `y_plan`/3. Also, another constraint states that `box1` has the same width of `first_floor`. The whole geometric model of this layout can be obtained by evaluating the expression:

```
basement_floor
```

We remember that the whole terraced building is obtained by starting from only two housing units, in the following denoted by `A` and `B`. The two units only differ for their second floors, where the unit type `A` do not contains the room `bed3`. This room instead is used by the designer to make a sort of bridge over the balcony when the units type `B` are used. First some reference variables are defined which are constrained to the values bounded to the variables used for the lower floors. Then two alternative layouts for the second floor are given, denoted as `second_floor_A` and `second_floor_B`, respectively:

```
DEF x_first = 945                              ;
DEF x_bed3  = x_balcony                        ;
DEF x_bath  = x_first - x_bed3 - x_stairwell ;
DEF y_bath  = y_plan - y_bed3
```

Before defining the first floors it is useful to take into consideration the flat floor under the roof, named `roof_floor_A` and `roof_floor_B`, which are two appropriately positioned instances of `L_space`:

```
DEF roof_floor_A =
 (T:<1,2>:<x_first,y_plan>~R:<1,2>:PI~L_space):
          <x_bath,x_stairwell,y_bed3, y_bath >;
DEF roof_floor_B =
 glue:(roof_floor_A(nw ^ ne)bed3)
```

So, we can define the `second_floor_A` layout plan as follows:

```
DEF second_floor_A = floor (se ^ se) bath
                           (ne ^ ne) bed2
                           (se ^ sw) terrace
WHERE
 floor = floor_base & roof_floor_A,
 bath  = L_space:<passage , x_bath - passage,
                    y_bath - passage , passage>,
 bed2 =(R:<1,2>:pi~L_space):<x_bath - passage,
          passage,y_bed1,y_bed3-y_bed1+passage>,
 terrace = O_space:<x_ground+x_balcony-x_first,
                                y_plan - y_bed1>
END
```

Notice that `floor` is obtained by intersecting `roof_floor_A` with `floor_base`. In order to define the layout plans of type "B" of first floor, it is then sufficient to add an instance of `bed3` to `roof_floor_A`:

```
DEF bed3 = O_space:<x_bed3,y_bed3>;
DEF second_floor_B =second_floor_A (nw ^ ne) bed3
```

1.3 *Schematic layout sections*

The 3D models of both type `A` and type `B` housing units are quite difficult to generate, mainly because (a) both 3D models contain several different sections and (b) such sections do not necessarily change in correspondence with internal partitions, so inducing very complex space models.

Hence, in order to automatically produce the set of space polygons which define the 3D schematic models of our housing units, in this paper a new generative method which starts from ordered sets of plans and sections is described. This method is derived by the intersection of extrusions, special case of the generalized product operation described in [3]. In that reference it is shown that the geometric model of the class of buildings with constant plan and section can be generated by pairwise intersecting all the cells of the 3D complexes produced by extruding both the plan and the section (2D complexes), after a proper embedding of them onto two coordinate subspaces of $\Re^3$.

In the case here discussed, we conversely have to combine *an ordered set* $\{P_1, \ldots, P_m\}$ of plans and *an ordered set* $\{S_1, \ldots, S_n\}$ of sections (see Fig. 3). The solution is found by computing $m \times n$ pairwise intersections of extrusions between an open subset of a plan and an open subset of a section. More formally, we can write that the result is generated as:

$$\bigcup_{i=1}^{m} \bigcup_{j=1}^{n} (P_i \cap O_j) \;\&\&\; (S_j \cap O_i)$$

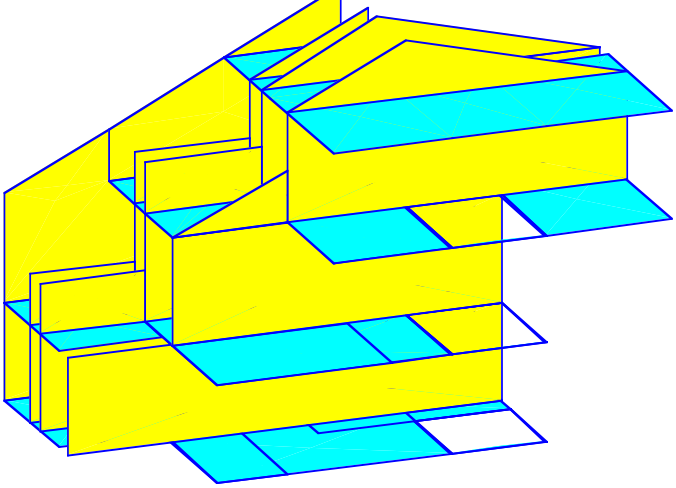


Figure 3: Plans and section of the building unit B, properly embedded in $\Re^3$.

where O_j, O_i denote the *open* stripe of "constant section" S_j and of "constant plan" P_i, respectively, and where && denotes the intersection of extrusions operator.

So, from the analysis of lateral views and sections of the building units, four zones "with constant section" are characterized, as it is shown in Figure 4. Such four sections have a similar structure and, if superimposed to the lateral view, lead to a partitioning in the regular cellular complex shown in Figure 5). The dimensions which define the grid underlying such a cellular decomposition are defined using the same variables which constraint the sizes of the plan elements, according to the following scheme:

```
DEF z_floor =300 %first and second floor heigth%;
DEF z_basement =z_floor*0.9 %basement  heigth  %;

DEF hs = h0-z_basement % level of basement     %;
DEF h0 = 0             % level of first floor  %;
DEF h1 = h0+z_floor    % level of second floor %;
DEF h2=h1+((h3-h1)*(l5-l4)/(l5-l3))%constraints%;
DEF h3=h1 + z_floor                % for the   %;
DEF h4=h1+((h3-h1)*(l5-l2)/(l5-l3))%roof slope %;

DEF l0 = 0 ;
DEF l1 = l0 + x_bed3 ;
DEF l2 = l1 + x_stairwell  ;
DEF l3 = l0 + x_first  ;
DEF l4 = l1 + x_ground  ;
DEF l5 = l1 + x_ground + x_terrace
```

According to the defined dimensions, the grid points used as reference for the sections can be computed by row starting from bottom, by using the function `section_grid`:

```
DEF section_grid =
  CAT:<<l1,l4,l5>          add_coord  hs ,
       <l1,l4,l5>          add_coord  h0 ,
       <l0,l1,l2,l3,l4,l5> add_coord  h1 ,
       <l2,l3,l4>          add_coord  h2 ,
       <l0,l2,l3>          add_coord  h3 ,
       <l2>                add_coord  h4 >
```

where the binary function `add_coord`, which adds the y coordinate to a sequence of real numbers to be interpreted as x coordinates of an aligned subset of grid points, is defined as follows:

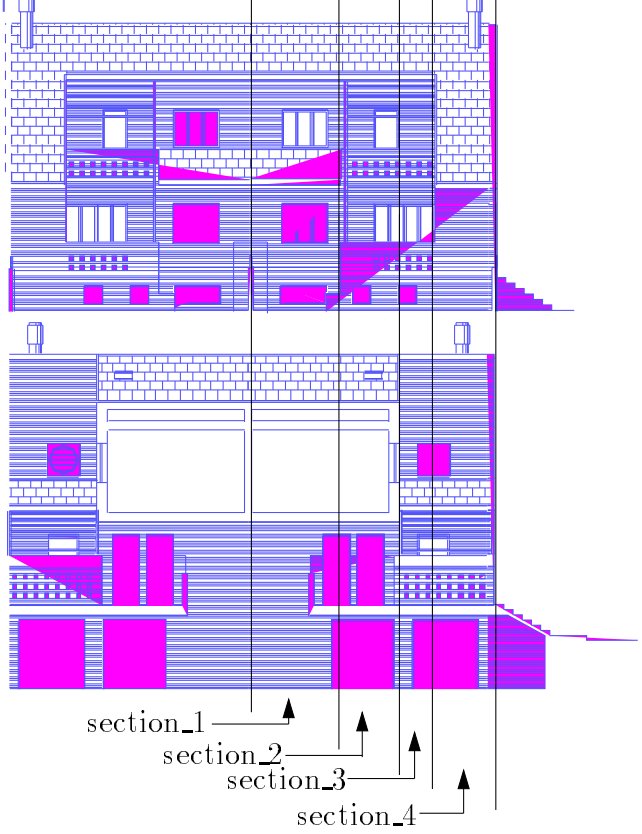


Figure 4: Zones "with constant section".

```
DEF add_coord (xx::IsSeq; y::IsReal) =
    AA:(AR~[[ID],K:y]):xx
```

Notice that, e.g.

```
<l1,l4,l5> add_coord hs=<<l1,hs>,<l4,hs>,<l5,hs>>
```

Starting from the given set of grid points, the four sections of the housing unit can then be defined simply by (a) selecting for each section the appropriate subset of grid points and (b) grouping the subset of points in order to the define the convex cells of the section and (c) grouping the convex cells to define some polyhedral cells (only when this is necessary), according to the usual semantics of the primitive function `MKPOL` (see Reference [6]):

```
DEF section_1 =
 MKPOL:<section_grid,
  <<1,3,4,6>,                      % basement     %
   <4,6,8,11>,<6,11,15,12>,<8,9,13>,%first floor%
   <9,11,15,18,17>,                % second floor %
   <17,18,19>>,                    % roof         %
   <<1>,<2,3>,<4>,<5>,<6>>> ;

DEF section_2 =  CAT_POL:<section_terr,
  MKPOL:<section_grid,
        <<1,3,4,6>,                % basement     %
         <4,5,8,11>,<8,9,13>,      % first floor  %
         <9,10,18,17>,             % second floor %
         <17,18,19>>,              % roof         %
        <<1>,<2,3>,<4>,<5>>>> ;

DEF section_3 =  CAT_POL:<section_terr,
 MKPOL:<section_grid,
        <<1,3,4,6>,                % basement     %
         <4,5,8,11>,               % first floor  %
```

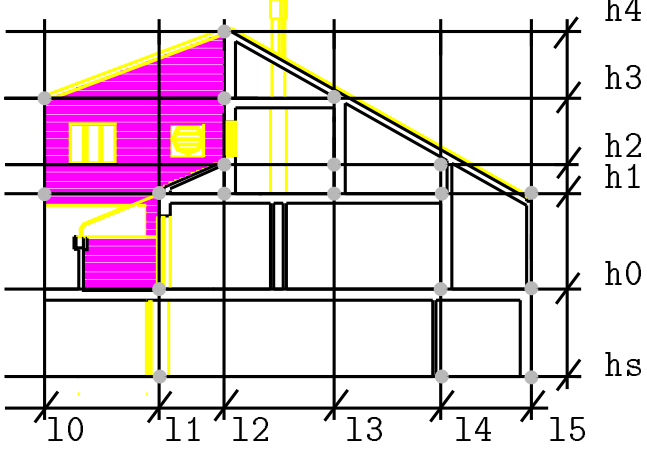


Figure 5: Basic grid of the sections definition.

```
        <7,10,18,16>,           % second floor %
        <16,18,19>>,            % roof         %
       <<1>,<2>,<3>,<4>>>> ;

DEF section_4 =
 MKPOL:<section_grid,
       <<1,3,4,6>,              % basement     %
        <4,5,8,11>,<10,11,14>,  % first floor  %
        <7,10,18,16>,           % second floor %
        <16,18,19>>,            % roof         %
       <<1>,<2,3>,<4>,<5>>>
```

The section of terraces, denoted as `section_terr`, is separately defined by making use of partially open rectangular polyhedra generated by the predefined function `OCUBOID`. In this way it is specified that the terraces are open spaces which are only partially bounded by polygons:

```
DEF section_terr =
   STRUCT:<T:<1,2>:<l3,h1>:
          (OCUBOID:<<1,1>,<1,0>>:<l4-l3,h2-h1>),
          T:<1,2>:<l4,h0>:
          (OCUBOID:<<1,1>,<1,0>>:<l5-l4,h2-h1>)>
```

The `OCUBOID` predefined function is applied first to a sequence of binary pairs, then to a sequence of equal length of reals. The length of both pairs defines the intrinsic dimension of the generated hypercuboid; the sequence of binary pairs defines which facets of such object must be closed and which must be open. E.g., `OCUBOID:<<1,1>,<1,0>>:<a,b>` defines a 2D rectangle (with area `a * b`) which is closed on all sides but the one defined by the highest value of the second coordinate.

1.4 *Combination of plans and sections*

In order to combine plans and sections, it is necessary to explicitly indicate what "zone" of each plan must be combined with a "zone" of a section. Consider, e.g., the layout plan of first floor. The various zones which interact with the four sections are selected by intersection with properly positioned *open* rectangular stripes. So, the widths of the various plan zones are defined according to the following variables:

```
DEF y_section_0 = -:delta          ;
DEF y_section_1 = 190              ;
DEF y_section_2 = x_stairwell      ;
DEF y_section_3 = y_section_2 + delta ;
DEF y_section_4 = y_plan + delta
```

The functions `sel_section_1`, `sel_section_2`, `sel_section_3` and `sel_section_4`, which respectively select the part (zone) of a plan to be combined with the sections `section_1`, `section_2`, `section_3` and `section_4`, are defined as follows:

```
DEF sel_section_1 =
   & ~[k:(stripe:<y_section_0,y_section_1>),id];
DEF sel_section_2 =
   & ~[k:(stripe:<y_section_1,y_section_2>),id];
DEF sel_section_3 =
   & ~[k:(stripe:<y_section_2,y_section_3>),id];
DEF sel_section_4 =
   & ~[k:(stripe:<y_section_3,y_section_4>),id]
```

where the function `stripe`, which returns the generic open stripe is given as:

```
DEF stripe (y1,y2::IsReal) =
   (T:2:y1:(OCUBOID:<<1,1>,<0,0>>:<10000,y2-y1>))
```

Analogously, some other functions must be defined which select the part of a section to be combined with a particular layout plan. With reference to the grid of Figure 5, it is easily verified that the section zone corresponding to the basement is constituted (`sel_included_pol`) by the elements between the y values `hs` and `h0`:

```
DEF sel_basement = sel_included_pol:<2,hs,h0>
```

At the same way it is possible to select the zones associated to the first floor (between `h0` and `h2`) and to the roof (between `h3` and `h4`):

```
DEF sel_first = sel_included_pol:<2,h0,h2>  ;
DEF sel_roof = sel_included_pol:<2,h3,h4>
```

In order to select the zone corresponding to the second floor it is instead necessary to use a different function (`sel_including_pol`), which select only the elements of a 2D section which contain the y stripe between `h2` and `h3`:

```
DEF sel_second = sel_including_pol:<2,h2,h3>
```

The two functions to select polyhedra which respectively are contained within or contain a stripe zone between `min_val` and `max_val`, with respect to the `coord` reference axis, are defined as follows:

```
DEF sel_including_pol (coord::IsIntPos;
                       min_val,max_val::IsReal)=
  (CAT~AA:(IF:<AND~[LE:min_val~min:coord~[ID],
                    GE:max_val~max:coord~[ID]],
               [id] , K:<>>)) ;

DEF sel_included_pol (coord::IsIntPos;
                      min_val,max_val::IsReal) =
  (CAT~AA:(IF:<AND~[GE:min_val~min:coord~[ID],
                    LE:max_val~max:coord~[ID]],
               [ID] , K:<>>))
```

Finally, the "generalized product" (see Reference [3]) `&&` is specialized as the function `sel_prod`, to be used to combine a zone of a plan with a zone of a section, where the two zones are selected by using the four functions previously discussed. Notice that `sel_prod` first by executing a generalized product of plans and sections properly embedded in $\Re^3$, then extract the 2D skeleton (`@2`), i.e. the set of boundary

polygons of all the cells, from the generated 3D complex:

```
DEF sel_prod (plan,sect::IsPair) =
   s1:sect:(s2:plan)
     (@2~&&:<<1,2,0>,<1,0,2>>) s1:plan:(s2:sect)
```

So, in order to combine the first floor plan with the first section we have to evaluate the PLASM expression:

```
<sel_first,first_floor>
  sel_prod <sel_section_1,section_1>
```

Since several pairs (plan, section) must be combined using the `sel_prod` operator, it may be convenient to define a new higher-level function, which systematically applies `sel_prod` to the proper argument pairs:

```
DEF combine_plans_sections =
   (STRUCT~AA:sel_prod~CAT~AA:DISTL~DISTR)
```

In this way the 3D models of two housing units type `A` and `B` which constitute (together with their reflected images) the whole building, are then given as

```
DEF unit_A = combine_plans_sections:
  <<<sel_basement,basement_floor>,
    <sel_first,first_floor>      ,
    <sel_second,second_floor_A>  ,
    <sel_roof,roof_floor_A>       >,
   <<sel_section_1,section_1>,
    <sel_section_2,section_2>,
    <sel_section_3,section_3>,
    <sel_section_4,section_4>>>;

DEF unit_B = combine_plans_sections:
  <<<sel_basement,basement_floor>,
    <sel_first,first_floor>      ,
    <sel_second,second_floor_B>  ,
    <sel_roof,roof_floor_B>       >,
   <<sel_section_1,section_1>,
    <sel_section_2,section_2>,
    <sel_section_3,section_3>,
    <sel_section_4,section_4>>>
```

1.5 *Definition of the staircase*

In this section the staircase structure is defined to be exactly positioned and matched with the empty space (stairwell) produced together to the automatically generated 3D models of the housing units.

The basic elements are a straight flight of stairs and a spiral stair with squared base. In order to define the first it is sufficient to put together a set of properly translated rectangular steps. To define the second we need to give also a triangular step, as well to connect the various steps with rotations and translations.

```
DEF S_flight (x,y,z::IsReal; n_steps::IsInt) =
  (STRUCT~##:n_steps):
       <((T:3:z~EMBED:1):(CUBOID:<x,y>)
        STRUCT (R:<2,3>:(PI/2)~EMBED:1):
        (CUBOID:<x,z>)),T:<2,3>:<y,z>> ;

DEF L_flight (x,y,z::IsReal;n_turns::IsIntPos) =
  (STRUCT~##:n_turns):
    <step1,T:3:z, step2,T:3:z, R:<1,2>:(PI/2)>
 WHERE
  step1 = (S:1:-1~T:1:(-:x)~T:3:z~EMBED:1
           ~S:<1,2>:<x,y>):(SIMPLEX:2)
   STRUCT (R:<2,3>:(PI/2)~EMBED:1)
                                :(CUBOID:<x,z>),
   STRUCT (S:<1,2>:<sqr2,sqr2>~R:<1,2>:(PI/4)
                ~R:<2,3>:(PI/2)~EMBED:1)
                                :(CUBOID:<x,z>),
  sqr2  = 2 ** 0.5
 END
```

The stairs from basement to first floor (`basement_stair`) and from first floor to second floor (`first_stair`), will be constructed by the basic elements so defined:

```
DEF basement_stair (x,z::IsReal) =
    (@2~STRUCT):<T:<1,2>:<X_step,Z_step+X_step>
       ,R:<1,2>:(pi/-2),final_step,
       (T:<1,2>:<dd,dd>~STRUCT~##:3):
                 <T:<1,2,3>:<dd,dd,Z_step>:flat,
       T:<1,2>:<dd,-:dd>:
             (S_flight:<X_step,Z_step,Z_step,1>),
       T:3:(3*Z_step),R:<1,2>:(pi/2)>,
       T:3:(z-delta):final_step>
 WHERE
  X_step = (x - Z_step)/ 2 ,
  Z_step = z / 9 ,
  dd     = Z_step / 2 ,
  final_step = S:2:-1:(EMBED:1:
               (CUBOID:<X_step+Z_step,X_step>)) ,
  flat   = L_flight:<X_step,X_step,Z_step,1>
 END;

DEF first_stair(x,y,z::IsReal;n_steps::IsIntPos)=
 (@2~STRUCT):<T:1:X_step,flight
            align:<<3,max,min>,<2,max,min>>turn,
            T:3:z:(EMBED:1:(CUBOID:<X_step,y>)),
            T:<1,3>:<-:X_step,z>,S:3:-1, flight>
WHERE
  n_step_flight = (n_steps - 4) / 2 ,
  X_step  = x / 2 ,
  Y_step  = (y - X_step) / n_step_flight ,
  Z_step  = z / n_steps ,
  flight  = S_flight:<X_step,Y_step,Z_step,
                                n_step_flight>,
  turn    = L_flight:<X_step,X_step,Z_step,2>
END
```

The total dimensions of each stair are given as input parameters `x`, `y` and `z` in their function definition. Notice that `y` is not given in the definition of `basement_stair` because it equates `x`. Notice also that a single `STRUCT` construct in the `staircase` definition put the `first_stair` above the `basement_stair`:

```
DEF staircase = STRUCT:
  <T:<1,2>:<x_bed3,y_plan-y_bed1>:
  (first_stair:<x_stairwell,y_bed1,z_floor,16>),
   T:<1,2,3>:<x_bed3,y_plan-x_stairwell,
                                -:z_basement>:
  (basement_stair:<x_stairwell,z_basement>)  >
```

At this point both the basic housing units can be completed by matching the staircase to the rest of the models: their generative functions can then be defined as follows:

```
DEF housing_unit_A = staircase STRUCT unit_A ;
DEF housing_unit_B = staircase STRUCT unit_B
```

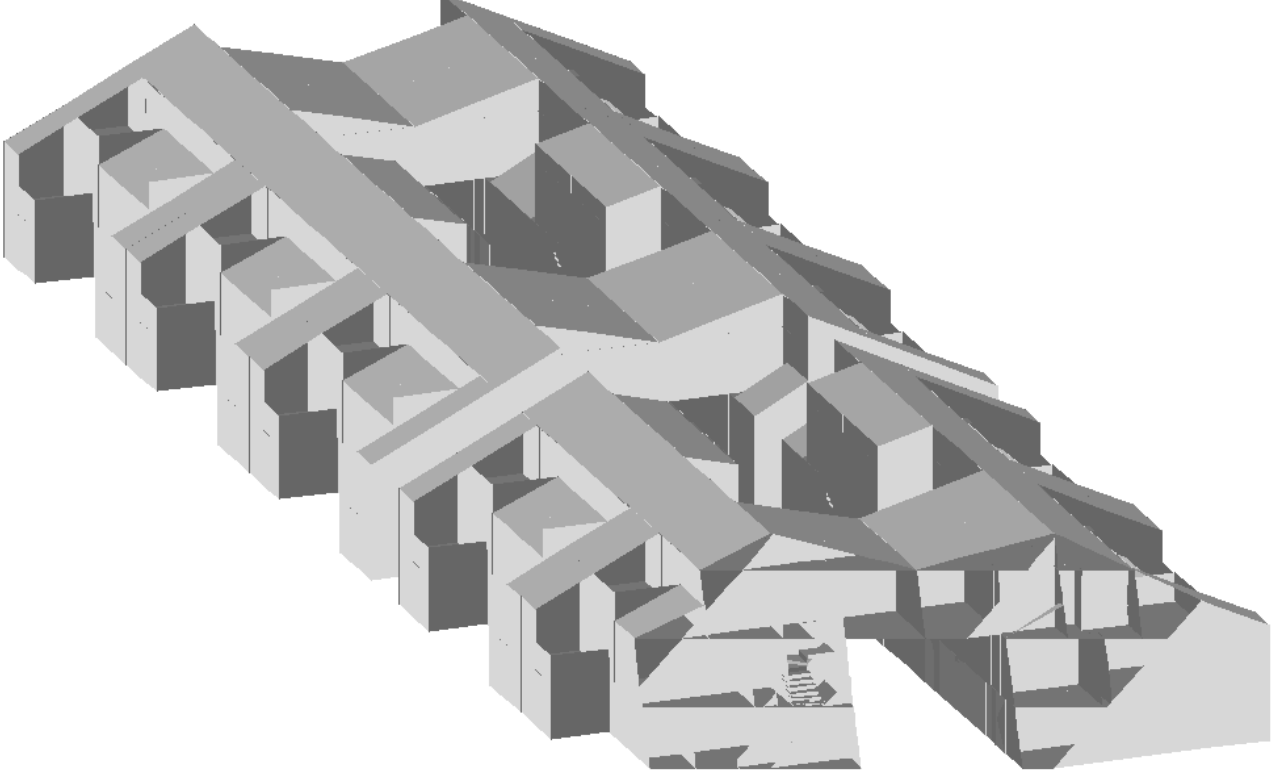


Figure 6: Complete terraced housing.

1.6 *Terraced building definition*

Now, starting from `housing_unit_A` and `housing_unit_B`, two groups of four instances of each unit are built, specularly reflected with respect to the axis x and y. So, the function `make_group` is defined to make such an arrangement. Also a global variable `DeltaY` is defined to furtherly aggregate such groups of buildings:

```
DEF DeltaY = 2 * y_plan;
DEF make_group = STRUCT~[ID,T:2:DeltaY~S:2:-1]
                       ~STRUCT~[ID,S:1:-1]
```

Finally, in order to generate the models of the such groups of four houses, it is sufficient to apply the function `make_group` to the two housing units:

```
DEF group_A = make_group:housing_unit_A ;
DEF group_B = make_group:housing_unit_B
```

The terraced building as a whole will then be given as an alternate series of instances of `group_A` and `group_B` (see Fig. 6) and of translations with parameter `DeltaY`:

```
DEF terraced_housing = STRUCT:
 < group_A,T:2:DeltaY,
   group_B,T:2:DeltaY,
   group_A,T:2:(DeltaY + 360),
   group_B,t:2:DeltaY,
   group_A >
```

References

[1] Backus, J., Williams, J.H. & Wimmers, E.L., 1990. An Introduction to the Programming Language FL. In *Research Topics in Functional Programming*, D.A. Turner (Ed.), Addison-Wesley, Reading.

[2] Backus, J., Williams, J.H., Wimmers, E.L., Lucas, P. & Aiken, A., 1989. *FL Language Manual, Parts 1 and 2*. IBM Research Report RJ 7100 (67163).

[3] Bernardini, F., Ferrucci, V., Paoluzzi, A. & Pascucci, V. 1993. A product operator on cell complexes. *Proc. ACM/IEEE 2nd Symposium on Solid Modeling and Applications*, J. Rossignac, J. Turner and G. Allen (Eds.), ACM Press, pp. 43–52.

[4] Howard,T.L.J., Hewitt,W.T., Hubbold, R.J. & Wyrwas, K.M., 1991. *A Practical Introduction to PHIGS and PHIGS PLUS*. Addison-Wesley, Reading, MA.

[5] Paoluzzi, A., Bernardini, F., Cattani, C. & Ferrucci, V., 1993. Dimension-Independent Modeling with Simplicial Complexes. *ACM Transactions on Graphics* 12:56–102.

[6] Paoluzzi, A., Pascucci, V. & Vicentino, M., 1995. Geometric Programming. A Programming Approach to Geometric Design. Accepted for publication on *ACM Transactions on Graphics*.

[7] Paoluzzi, A. & Sansoni, C., 1992. Programming Language for Solid Variational Geometry. *Computer Aided Design*, 24:349–366.