Declarative procedural generation of architecture with semantic architectural profiles

Levi van Aanholt and Rafael Bidarra
Computer Graphics and Visualization Group
Delft University of Technology
The Netherlands
l.s.vanaanholt@student.tudelft.nl, r.bidarra@tudelft.nl

***Abstract*—Procedural content generation (PCG) for architecture is widely used in a variety of digital media, most notably in games. However, such methods are often limited in their expressive range, and require considerable technical knowledge to create non-trivial architectural structures. We present a novel tile-based PCG approach for generating architecture, that proposes the use of *architectural profiles*, a declarative characterization of architectural typology, within a generic tile solving framework. An architectural profile consists of a set of tiles, representing atomic architectural building blocks, and a set of declarative constraints and rules, specifying which conditions a tile configuration has to satisfy to be valid. These conditions are translated into logic constraints, and used by a tile solver to control tile placement in a bottom-up manner. Eventually, each valid model output by the solver is a representative instance of its architectural profile. We describe an implementation of this approach with Answer Set Programming, using an off-the-shelf constraint solver for model generation. We performed an expressive range analysis, and concluded that our declarative method is quite controllable and can be steered over a broad range of architectural structures, regarding density and repetitiveness. Due to this expressive range and control, our tile-based method is very suitable for the customized development of urban environments for games.**

***Index Terms*—Procedural Content Generation, Architecture, Expressive Range, Tile Solving**

I. Introduction

Procedural content generation (PCG) for architecture is increasingly used in a variety of digital media, allowing for the cost-effective creation of urban spaces for ads, movies and games, and reducing the volume of hand-made content needed [1].

Architectural typology categorises buildings based on their physical and functional properties, and this characterization can be applied to PCG for architecture. We say a PCG technique is *declarative* when the generator can be easily steered to generate models of the intended typology [2]; and it is *comprehensive* if it is capable of generating a variety of typologically distinct models [3]. Many current PCG approaches use grammars, which are highly configurable and fast, but are difficult to write and apply, thus lacking in declarative control. Other approaches are data-driven, which makes them easier to use, but their expressive range is dependent on the existence of good training sets, potentially compromising comprehensiveness.

We present a novel tile-based PCG approach for generating architecture that is both declarative and comprehensive. Our method does not rely on grammars or a training set, but on the definition and use of *architectural profiles*, a semantic characterisation of architectural typology suitable for use within a generic tile solving framework (Section III). This declarative approach is easily configurable (Section IV), and able to generate a broad range of architectural structures (Section V).

II. Related Work

Previous research results fall into two categories: tile solving methods and procedural modelling for architecture.

A. Tile Solving

Tile solving deals with the challenge of filling a space with tiles given a tile set, under a variety of neighborhood constraints for each tile, e.g. allowing only a tile subset at its adjacent positions. Wang Tiles approaches [4] formalize this problem. A common data-driven PCG approach with tiles is tile synthesis, consisting of a tile retrieval step and a tile solving step. This approach achieved successes in texture synthesis [5], [6] and in 3D model synthesis [7]; the latter takes a 3D object as input, splits it into tiles using a grid, and proceeds to do tile solving with a greedy algorithm. Tiles can also be obtained using offset statistics [8]. Model synthesis was later generalized by using Markov Random Fields [9], which allow for statistical constraints over larger tile groups. WaveFunctionCollapse (WFC) [10] is another tile solving method, which omits a tile retrieval step. It has been popular in the PCG community and has also inspired valuable research work [11], [12]. In particular, WFC can be solved using Answer set Programming (ASP) [13], a declarative language for computationally complex problems. This approach is promising because by using ASP it is exact and extendable.

B. Architectural Semantics in PCG

Architecture can be understood as the meaningful organization of space, accomplished by purposefully placing construction elements. PCG techniques model architecture by connecting architectural semantics with computational means. Shape grammars with split rules [14] model architecture as a hierarchic application of refinements to a base shape. This approach successfully models building envelopes and classic architecture [15] but does not easily capture the overall building semantics. To create believable buildings, multiple distinct generators can be orchestrated [2], or a building layout can be optimized using a trained network [16]. Grammars representing diverse architecture can be blended by adding labels [17], creating a large variety of new mixtures. Moreover, labeling shapes as public and private space [18] results in evocative spatial layouts within a hierarchical generation process.

III. Approach

We describe our novel tile-based approach for the generation of architectural structures, that proposes the notion of *architectural profile* and its use within a generic tile solving framework. An architectural profile is a declarative characterisation of a given architectural typology, and consists of a set of tiles (as atomic architectural building blocks) and a variety of declarative constraints and rules (as validity conditions for tile configuration).

Before describing in detail the components of the architectural profile, we first introduce the basic elements of a tile solving framework.

A. General Tile Solving Framework

The essential elements of a general framework for tile solving can be summarized around the notion of Tile Constraint Profile, as follows:

$$TileConstraintProfile = (T, A, R), \tag{1}$$

where T is the set of available tiles, A a set of adjacency conditions between tiles, and R a set of validity constraints. Figure 1 illustrates the global approach: given a Profile and the available input space, the tile solver incrementally 'fills that space' with tiles that comply to all conditions and constraints expressed, yielding a valid output model.

For this, we consider the input space subdivided in equally sized cells c, each cell susceptible to hold one and only one tile. The goal of the solving process is to assign one tile to each cell without violating the profile conditions and constraints, in A and R.

1) Tiles: T is a set of 3-dimensional tiles, each of which has a bounding box of the size of a (space) cell c. Some examples of tiles are given in Figure 2, fitting a cell size of $5 \times 5 \times 4$. For convenience (and unlike Wang tiles), we will consider that a tile may be rotated (by multiples of) 90° around the vertical axis (Z). This allows for a strong reduction in the size of the tile set T, required for generating moderately complex structures.

2) Adjacency conditions: A is the set of Adjacency Conditions, i.e. hard constraints denoting each pair of tiles (t_1, t_2) that may be assigned to adjacent cells, along some direction d; this constraint is indicated as $t_1 \xrightarrow{d} t_2$. An adjacency condition is, by definition, always bidirectional, i.e. $t_1 \xrightarrow{d} t_2 \iff t_1 \xrightarrow{-d} t_2$. Moreover, an adjacency condition $t_1 \xrightarrow{d} t_2$ also holds for all 90° rotations (mentioned above) equally applied to the two tiles. In short, during the solving process, two tiles can only be placed adjacent to each other if they comply to some adjacency condition of A.

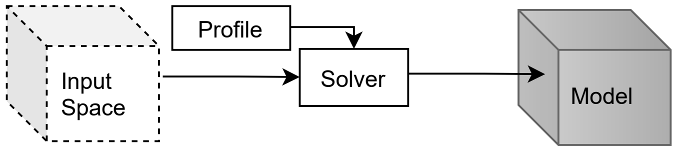


Fig. 1: An input space and a profile are input to the solver yielding an output model.

3) Validity Constraints: R is the set of Validity Constraints, i.e. profile rules that additionally constrain the solving process, by specifying explicit restrictions on tile placement. Constraints can be either hard constraints, which have to be satisfied by all tiles in the output Model, or soft constraints, each of which poses a penalty cost when violated.

An example of a hard validity constraint would be to require the output to follow a valid Sudoku pattern, i.e. for any row, column or sub grid in the Model, no two cells may contain the same tile. An example of a soft validity constraint is optimizing for a uniform distribution of tiles in the Model. The penalty cost can be described as the variance of the occurrences of the tiles in the Model.

Summarizing, the purpose of the whole solving process for a Tile Constraint Profile (T, A, R) consists of assigning a tile from T to each and every cell of the input Space, yielding an output Model such that (i) every tile satisfies the adjacency conditions in A, (ii) it satisfies all hard validity constraints in R, and (iii) it minimizes the cost penalty of all soft validity constraints in R.

B. Architectural Profile

An Architectural Profile is a semantic specification tailored to declaratively constrain a tile solver for architectural generation. The specification for an architectural profile applies the general framework of the previous subsection to an architectural context: tiles represent architectural elements, each with its own semantics, and are combined to create architectural structures called *shapes*, which in turn can be further combined. The essential solving process of architectural profiles will, therefore, focus on the placement of shapes in the input space, rather than on tile placement. An architectural profile is defined as:

$$ArchitecturalProfile = (T_a, A, S, A_S, R_S), \tag{2}$$

where T_a is the set of tiles with architectural semantics, representing architectural building elements and their function, and A is a set of adjacency conditions between tiles. S is the set of profile shapes, built with tiles of T_a, A_S is a set of adjacency conditions between shapes of S, and R_S is a set of architectural validity constraints on shapes of S.

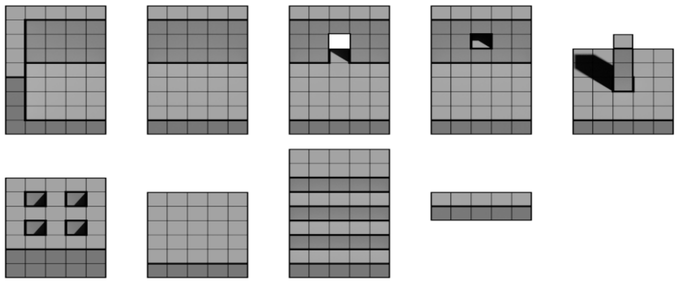


Fig. 2: Tiles used in the profiles showcased in this paper. From left to right, first row (interior tiles): corner, wall, door, window, interior area; second row (exterior tiles): roof, street, stairs, landing, void tile. Stairs and landing tiles can be both interior and exterior.

1) Tiles: In an architectural profile each tile has some architectural meaning, or semantics e.g. walls, windows and doors. Moreover, semantics on tiles can be captured in the form of labels. For the examples in this paper, we will make use of two tile labels to indicate that a tile is meant for building *interior* or *exterior* (or possibly both). This tile architectural semantics is illustrated in Figure 2, where each tile corresponds to an architectural element, and has one or more labels. Enriched with this semantics, tiles can be grouped and filtered, defining *types* which will prove convenient for describing shapes.

2) Adjacency conditions: Semantics is also defined on the adjacencies between tiles, expressing the meaning of how tiles may relate to each other. This can define, for example, an allowed navigation direction from one tile to the other (*traversal* condition), or whether a tile supports construction above it (*construction* condition). These meaningful adjacencies can be defined between a tile and a whole type of tiles, yielding so-called *typed adjacency conditions*: $t \xrightarrow{d} type \in A_{type} \subseteq A$. Figure 3 exemplifies the two typed adjacency conditions mentioned above, used throughout the examples in this paper. As before, typed adjacency conditions define in which direction(s) they hold. Typed adjacency conditions are useful to specify which building elements can be connected within a shape.

Finally, we define a typed adjacency condition as an 'entrance', when it involves an entrance tile t (e.g. a 'door') and (tiles of) some *type* (e.g. 'traversal'), along the entrance direction d; it is denoted as $t \xrightarrow{d,e} type$. Entrances defined in this way are useful as the entry point for shapes to be connected together, thus allowing for the definition of shape connectivity.

3) Shapes: The central concept of an architectural profile is a *shape*, defined as a *connected* cluster of tiles, all sharing a common label (e.g. *interior*). The term 'connected' here has the common recursive definition (i.e. t_1 is connected to t_2 if either t_1 is adjacent to t_2 or t_1 is adjacent to some tile t_i that is connected to t_2), but with the condition that these adjacencies are of the same type (e.g. *traversal*), and are not entrances. All tiles that are connected in this way belong to the same

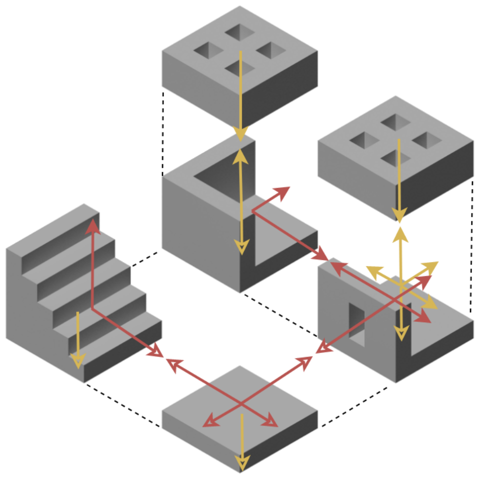


Fig. 3: Semantics on tile adjacency conditions, in two types: *traversal* (in red) and *construction* (in yellow). The interior here, composed of a corner tile and a door tile, allows for *construction* above them, as indicated by the upward arrows, which are matched by the roof tiles above, with downward arrows. The door tile, allowing for a *traversal* entrance adjacency condition (towards the exterior), matches with the street tile, providing access to the interior.

shape. This adjacency type is, therefore, essential in the shape definition.

Because we are interested in the generation of architecture, shapes can be seen as the larger structures (e.g. streets or buildings), and the tiles as the basic elements composing that structure (see Figure 4).

As an architectural structure, a shape can be expected to assume a variety of sizes. Depending on the shape and on its intended architectural use, some way has to be provided to indicate the valid range(s) for a shape's size. For simplicity, we will assume here two simple bounding boxes, indicating the shape's minimum and maximum sizes, respectively.

Summarizing, in order to generate a shape, a cluster of connected tiles has to be assembled, such that (i) they all share the same specific label, (ii) their adjacencies are of the same type, and (iii) the tile cluster fits within the given size range.

4) Shape adjacency conditions: Similar to adjacency conditions defined between tiles, we define adjacency conditions between shapes to constrain their placement. Adjacency conditions between tiles described local placement relations

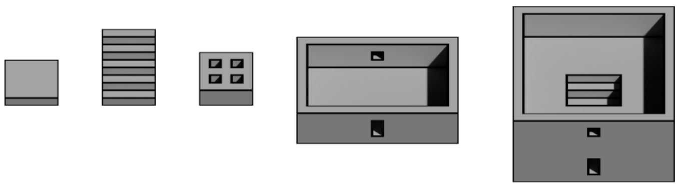


Fig. 4: Shapes used in the profiles showcased in this paper: (from left to right) street, stairs, roof, one-story house, two-story house.

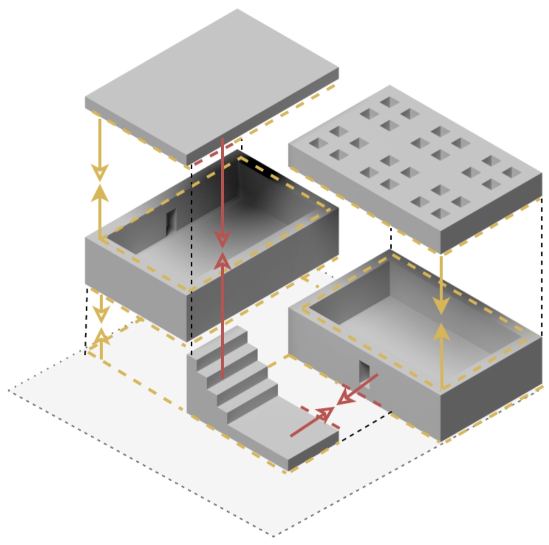


Fig. 5: Shape adjacencies between shapes, indicated by the arrows, traversal in red and construction in yellow. Each shape adjacency can only exist if there is a corresponding shape adjacency condition defined in the profile.

between architectural elements such as walls, ceilings and stairs. Adjacency conditions between shapes describe placement relations between architectural structures. For example, a building and its connection(s) with the street or with other buildings, illustrated by figure 5.

We say that two shapes, $shape_a, shape_b \in Model$, are adjacent over a given adjacency *type* when they include two adjacent tiles, t_a, t_b, such that $t_a \xrightarrow{*} t_b \in A_{type}$, where the * symbol stands for any direction. An adjacency condition between the two shapes is then indicated as $Shape_a \xrightarrow{*} Shape_b \in A_{S,type}$; and, because tile adjacencies are symmetric relations, so are shape adjacencies. We will, therefore, simply indicate a shape adjacency condition as $Shape_a \overset{*}{\leftrightarrow} Shape_b \in A_{S,type}$.

All adjacent shapes in the model have to satisfy the shape adjacency conditions.

5) Architectural validity constraints: Architectural validity constraints are important to express generic conditions that hold for most structures. In addition, they help steering the generation process towards the domain of plausible and feasible architectural models. For the examples in this paper, we define the following three rules, controlling traversability, gravity and occurrence.

Traversability This constraint aims at ensuring that the model is traversable, i.e. that all shapes in it are accessible, via adjacencies of type *traversal*. Without this, some architectural structure might never be reachable from the remaining ones. In formal terms, this means that the graph defined by the tile adjacency conditions of type *traversal* is connected.

Gravity In most architectural structures, each construction always stands or leans on some previous structure, possibly including the ground, rather than just floating above it. The gravity constraint aims at ensuring the connection of each shape to the ground, which can be itself regarded as a shape on its own. In formal terms, this requires that all graphs defined by the tile adjacency conditions of type *construction* are connected with the ground.

Occurrence The occurrence constraint allows for the specification of a desired density for a given shape in the model. In this way, one can control, for example, the density of house placing, ranging from, say, a rural village to a dense social neighborhood. It also allows to prioritize the occurrence of some shapes over others.

Summarizing, the complete solving process for an architectural profile (Eq. 2) consists of successively generating shape instances, and 'attaching' them to the model, by finding for them a location and orientation that (i) adheres to all shape adjacency conditions, and (ii) fulfills all validity constraints defined in the profile.

IV. Evocative Examples

In this section, we present several examples of architectural profiles that are illustrative of the power of the concept. For this, we first define a simple architectural profile that forms the basis for all subsequent variants.

Our Base Profile uses the tile set T_S shown in Figure 2, and includes the Traversability and Gravity validity rules for all building constructions. The adjacency conditions for this Base Profile A are obtained from one example model, by registering all tile adjacencies found between its tiles.

The Base Profile employs the same shapes shown in Figure 4. Additionally, other derived profiles also use the 'high-rise' and the 'monolith' shapes, which represent larger buildings. Following the shape definition in the previous section, a shape is defined by specifying a label, an adjacency type and a bounding box. For example, a small one-story house can be defined with [traversal, interior, (3,2-3,1)], while other bigger one-story buildings can use larger values for the bounding box. In these profiles, Shape Adjacencies have a type, either traversal or construction which is indicated by τ and κ.

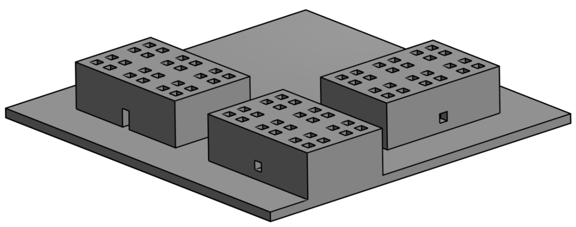
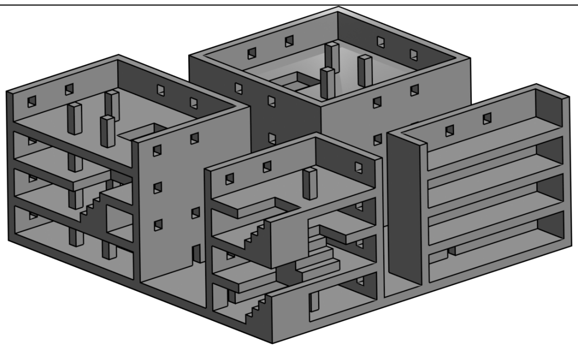
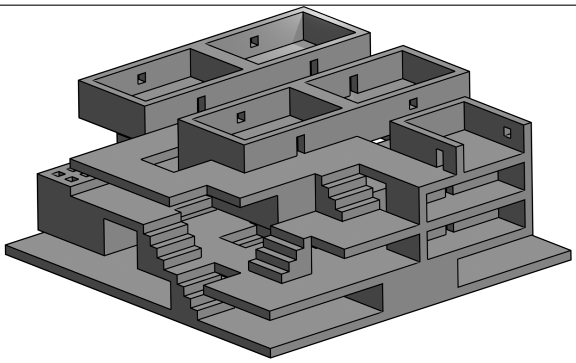
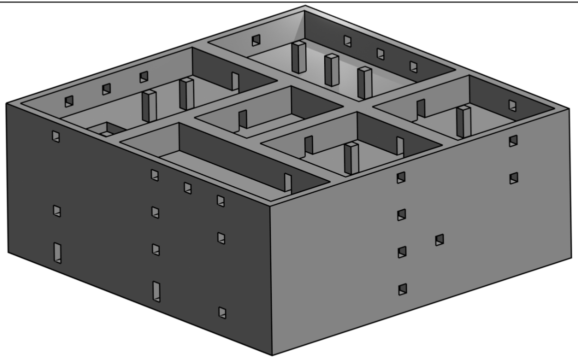
A. Examples

Table I presents four examples of architectural profiles, all of them extending the Base Profile above. For each of these profiles, we depict one model output by the solver for it.

1) One-story Profile: the street, one-story house and roof shapes are used to create a flat composition that is only traversed horizontally. The street is connected with shape adjacencies to itself and the one-story shape through the traversal. One-story shape is connected with the street, the roof and the ground through construction.

2) One-story Stacked Profile: the inclusion of a shape that allows for vertical traversal can dramatically change the generated models. We modify the [One-story] profile to become the [One-story] [stacked] profile. The stairs shape is added and it is connected using shape adjacencies with the street and itself through traversal, and with the ground through construction. Also a one-story house is connected to

TABLE I: Evocative examples

 [One-story]	 [High-rise]
 [One-story][stacked]	 [Monolith]

[One-story]
$S = \{$
$street, 1story, roof\}$
$S_A = \{$
$street \xleftrightarrow{\tau,*} 1story,$
$street \xleftrightarrow{\tau,*} street,$
$roof \xleftrightarrow{\kappa,*} 1story,$
$street \xleftrightarrow{\kappa,*} ground,$
$1story \xleftrightarrow{\kappa,*} ground,$
$\}$

[One-story][stacked]
$S =$ *([One-story]).*S + $\{stairs\}$
$S_A = \{$
([One-story]).S_A + $\{$
$stairs \xleftrightarrow{\tau,*} street,$
$stairs \xleftrightarrow{\tau,*} stairs,$
$stairs \xleftrightarrow{\tau,*} ground,$
$1story \xleftrightarrow{\kappa,*} 1story,$
$street \xleftrightarrow{\kappa,*} 1story$
$\}$

[High-rise]
$S = \{$
$street, highrise, roof\}$
$S_A = \{$
$street \xleftrightarrow{\tau,*} highrise,$
$street \xleftrightarrow{\tau,*} street,$
$roof \xleftrightarrow{\kappa,*} highrise,$
$street \xleftrightarrow{\kappa,*} ground,$
$skyscraper \xleftrightarrow{\kappa,*} ground$
$\}$

[Monolith]
$S = \{$
$street, monolith, roof\}$
$S_A = \{$
$street \xleftrightarrow{\tau,*} street,$
$street \xleftrightarrow{\tau,*} monolith,$
$roof \xleftrightarrow{\kappa,*} monolith,$
$street \xleftrightarrow{\kappa,*} ground,$
$monolith \xleftrightarrow{\kappa,*} ground$
$\}$

itself through construction. This results in the buildings being stacked on top of each other. Because stairs exist, walkways are being generated above the ground to connect the stacked buildings with each other.

3) High-rise Profile: the [High-rise] profile is a modification of [One-story] profile, substituting the one-story shape for the much larger high-rise shape. This modification makes buildings multi-storied and increases the building volume.

4) Monolith Profile: in the [Monolith] profile, the monolith shape is used that takes all the volume of the input space. This results in the model consisting of one large building.

V. Expressive Range analysis

We evaluate our technique by analysing the expressive range [19]. This evaluation technique analyses the variety a generator can produce and the configuration effort of a PCG technique. It has been used to analyse many PCG techniques such as a road generator [20] and several learned generators of 2D game levels [21]. For our purposes, we define two metrics, density and repetitiveness, and evaluate how the output of various profiles falls within these metrics.

A. Density

Density is a common metric in architecture, indicating the ratio between interior and exterior space. We classify interior space as that of shapes that form an enclosed space, and we count the tiles in these enclosed shapes ($interior$). Exterior space then are (the tiles of) all remaining shapes ($exterior$). The density is given by $density = interior/(interior + exterior)$

B. Repetitiveness

The repetitiveness metric indicates the prevalence of some patterns in a model. We focus only on the repetitiveness that occurs through the placement of buildings, i.e. regarding interior tiles. First the repetitiveness of all relevant tiles is calculated individually. We define the repetitiveness of an individual tile with a specific rotation as the ratio between r, the highest amount of times it is repeated a specific distance apart and r_t, the total occurrence of that rotated tile in the model. The repetitiveness for the entire model is calculated as the weighted average of the individual repetitiveness for all (interior) tiles, where each weight is the normalized occurrence of the corresponding rotated tile in the model.

C. Model generation

We construct twelve profiles for the analysis. A maximum of 100 model samples are generated per profile. Each model is of size $8 \times 8 \times 4$ cells. We use a concise naming scheme, previously used in the evocative examples section, to describe the different profiles. This convention indicates the building shapes used and additional other designations that alter the profile. The [stacked] designation means that buildings are allowed to stack on top of each other and the stairs shape is added to allow for vertical traversal. When [sparse] or its opposite [dense] is used, an occurrence rule is used on the building shapes in the profile, [sparse] indicating one building shape in the Model space in total, and [dense] indicating at least one building shape in every equal quadrant of the Space. [adjoined] means that separate building shapes can directly be connected door to door. Lastly, [naor] stands for no access on roof, meaning the top of buildings is never walkable (which they otherwise are).

D. Exploring the expressive range

Our metrics result in a two dimensional evaluation space with noteworthy properties shown in Table II. We now discuss each quadrant of this space.

1) Lower left: This quadrant indicates models that are varied and are sparsely built. The corner point is the empty model, no density and therefore no possibility for patterns. Profile a and e, that can only build on the ground and are sparsely built, naturally reside in this space. Also many profiles that stack buildings on top of each other are placed in this quadrant, f, g and h. These profiles are varied due to the existence of a lot of possibilities to stack buildings in a disordered way. To connect these freely placed buildings, a large amount of walkways are placed, limiting the density.

2) Upper left: This quadrant contains repetitive sparsely built models. Occupied by profile b, c and i. No models exist for the corner point, because our repetitiveness metric is zero if the density is zero.

3) Lower right: This quadrant has dense models with little repetitiveness. In the profiles that we have included in this investigation, one cannot find such models; these would need more tiles than the present limited tile set of 10 tiles, which does not provide a large enough variety of combinations.

4) Upper right: This quadrant is dense and highly repetitive. This is occupied by models from high-rise(k) and monolith(l) profiles, as these fill much space with repetitive buildings elements. The repetitiveness here stems predominantly from vertical placement patterns.

Looking closer at the impact of small changes between profiles, the following observations can be made on the changes in the expressive range.

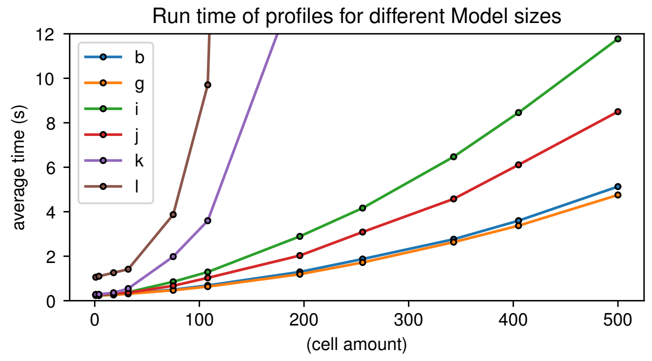


Fig. 6: Runtime analysis of the implementation. The different colored plots correspond to different profiles from Table II. Every data point is the average runtime of 10 model generations. The size of the shapes in the profile has a large impact on runtime: profiles that place bigger shapes, such as high-rise shapes and monolith shapes in profile k and l, take much longer to solve than profiles with small one-story buildings, such as profile b.

Profiles d relative to g show the impact of the [naor] designation to disallow traversal on the top of buildings. d's Profile is more constrained and a subset of g's Profile, but the samples of g mostly stay out of the area covered by d's samples instead of covering the whole space evenly. This suggests for profile g that model distribution is biased towards low repetitiveness. The declarative control added by [naor] in profile d is useful to create higher repetitive models.

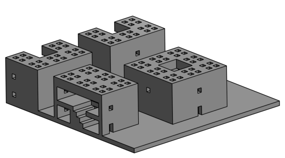
The sample distributions for profiles b and c are roughly the same, although c is a subset of b. At the same time the lower repetitive models in a are not covered in b. This suggests that b's profile model distribution is biased to be dense.

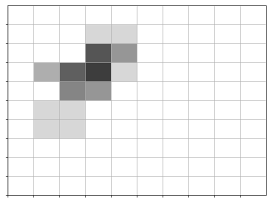
VI. Implementation

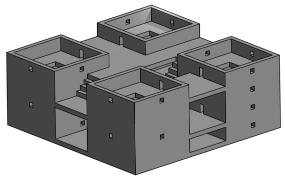
We have used Answer Set Programming to implement architectural profiles. This implementation consists of a dynamic part, that converts the input space and an architectural profile to ASP, and a static part, consisting of fixed logic written in ASP to constrain the Model to architectural profile adherence (described in Section III). These two parts are combined and given to an ASP solver, in our case, the off-the-shelf solver clingo [22]. The solver then outputs a model that adheres to the architectural profile. For every new model generated the solver is restarted with a new random seed.

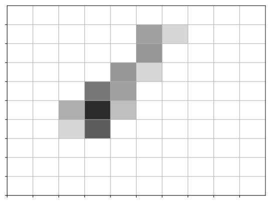
Solving an architectural profile is a complex problem, as it is generally as hard as tile solving (which is proven to be NP-hard [23]). Evidence of this complexity is also given in Figure 6, which shows an exponential run-time growth with Model size. To make solving an architectural profile computationally feasible we solve shapes and profiles separately. First, given the shape definitions in a profile, a finite number of shape instantiations are generated and stored. Subsequently these are made available in the profile solver to build models with.

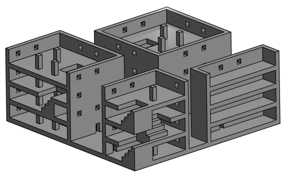
TABLE II: Expressive range analysis of twelve architectural profiles: above each bin plot histogram, one of its sample models is shown. Lower right corner accumulates all expressive range results (color indicates the amount of models in a bin).

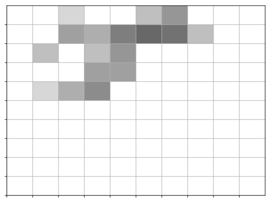

i. [Two-story] [dense]

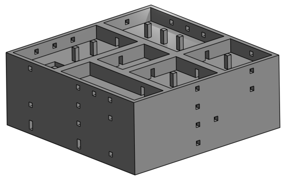


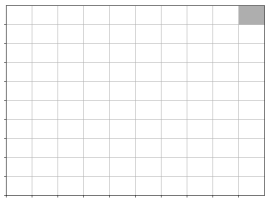

j. [Two-story] [dense] [stacked] [naor]

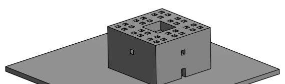


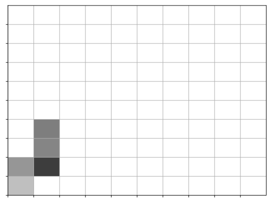

k. [High-rise]

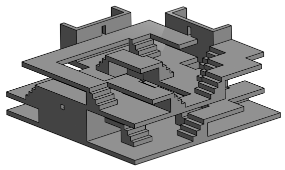


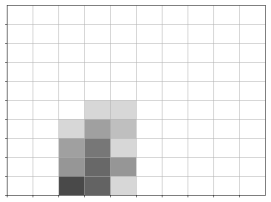

l. [Monolith]

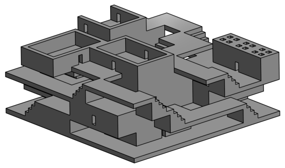


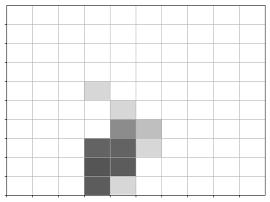

e. [Two-story] [sparse]

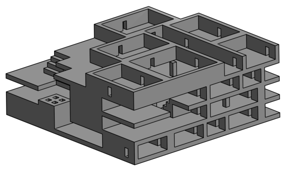


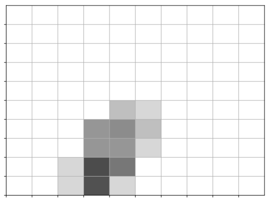

f. [One-story] and [Two-story] [stacked]

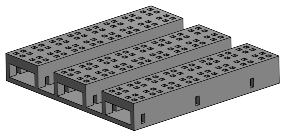


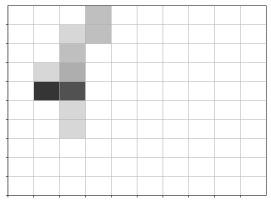

g. [One-Story] [stacked] [dense]

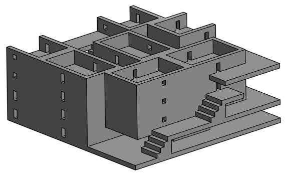


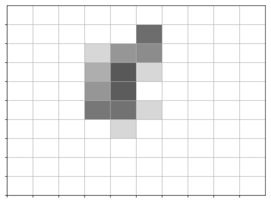

h. [One-story] [stacked] [adjoined]

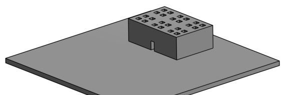


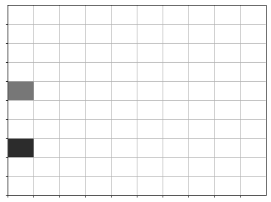

c. [One-story] [dense]

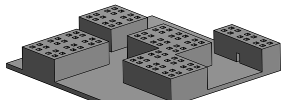


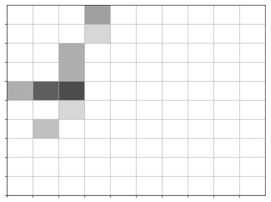

d. [One-story] [stacked] [naor] [dense]

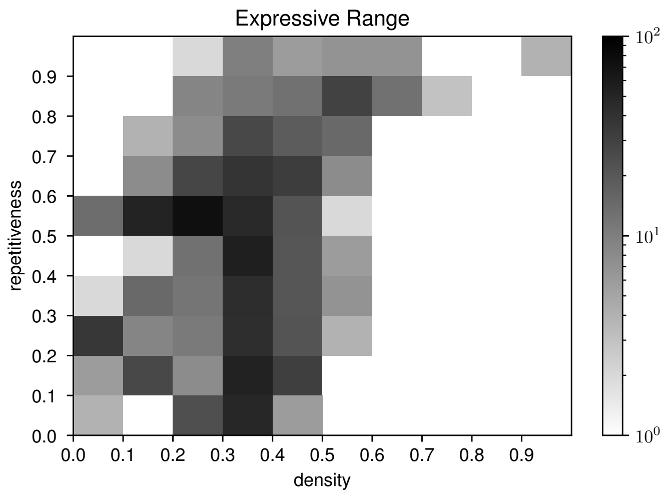



a. [One-story] [sparse]




b. [One-story]





Additionally, we keep low the amount of unique shapes used in the profile solver: two shape instantiations per shape definition. Our ASP implementation only makes use of hard constraints (no soft constraints were used). Finally, the current implementation is not optimal, and can be improved by addressing all performance bottlenecks, exploring further use of solving heuristics in clingo, and making use of multi-shot solving within clingo [24].

VII. Conclusion

We present a novel tile-based PCG method for generating architecture. The method is centered on the notion of *architectural profiles*, a semantic specification that declaratively characterizes architectural typology. By combining a set of tiles, meaningful adjacency conditions among them, and a variety of validity constraints, architectural profiles offer a powerful vocabulary for steering a 3D tile solver towards the generation of many creative architectural structures. From our expressive range analysis, we have shown that this method is tune-able to generate a large variety of architectural structures over significant ranges of density and repetitiveness.

Our present implementation in Answer Set Programming, is serviceable as an offline generator with modest to moderate model sizes. We believe that this method has a large potential for generating urban environments in (indie) game contexts, including the Minecraft concept [25]. We are currently exploring its application to the generation of narrative-rich urban settlements [26].

Interesting future work includes (i) automating the creation of an architectural profile by using machine learning on example data, (ii) recursively defining architectural shapes within profiles, further expanding the expressive range of the method, and (iii) the development of a designer GUI for allowing the interactive configuration of architectural profiles.

Acknowledgment

We thank Adam Smith for helping us improve our solver performance with his ASP and clingo expertise.

References

[1] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Beneš, "A survey on procedural modeling for virtual worlds," *Computer Graphics Forum*, vol. 33, no. 6, pp. 31–50, 2014, doi: 10.1111/cgf.12276.

[2] T. Tutenel, R. M. Smelik, R. Lopes, K. J. de Kraker, and R. Bidarra, "Generating consistent buildings: A semantic approach for integrating procedural techniques," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 274–288, Sep. 2011.

[3] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. de Kraker, "The role of semantics in games and simulations," *ACM Computers in Entertainment*, vol. 6, pp. 1–35, 2008.

[4] H. Wang, "Proving theorems by pattern recognition — ii," *The Bell System Technical Journal*, vol. 40, no. 1, pp. 1–41, Jan 1961.

[5] M. F. Cohen, J. Shade, S. Hiller, and O. Deussen, "Wang tiles for image and texture generation," in *ACM SIGGRAPH 2003 Papers*, ser. SIGGRAPH '03. New York, NY, USA: ACM, 2003, pp. 287–294. [Online]. Available: http://doi.acm.org/10.1145/1201775.882265

[6] A. Lu, D. S. Ebert, W. Qiao, M. Kraus, and B. Mora, "Volume illustration using Wang cubes," *ACM Trans. Graph.*, vol. 26, no. 2, Jun. 2007. [Online]. Available: http://doi.acm.org/10.1145/1243980.1243985

[7] P. Merrell, "Example-based model synthesis," in *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, ser. I3D '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 105–112. [Online]. Available: https://doi.org/10.1145/1230100.1230119

[8] X. Wu, C. Li, M. Wand, K. Hildebrandt, S. Jansen, and H. Seidel, "3d model retargeting using offset statistics," in *2014 2nd International Conference on 3D Vision*, vol. 1, Dec 2014, pp. 353–360.

[9] Y.-T. Yeh, K. Breeden, L. Yang, M. Fisher, and P. Hanrahan, "Synthesis of tiled patterns using factor graphs," *ACM Trans. Graph.*, vol. 32, no. 1, Feb. 2013. [Online]. Available: https://doi.org/10.1145/2421636.2421639

[10] M. Gumin, "WaveFunctionCollapse," 2016. [Online]. Available: https://github.com/mxgmn/WaveFunctionCollapse/

[11] A. Sandhu, Z. Chen, and J. McCoy, "Enhancing Wave Function Collapse with design-level constraints," in *Proceedings of the 14th International Conference on the Foundations of Digital Games*, ser. FDG '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3337722.3337752

[12] I. Karth and A. M. Smith, "Addressing the fundamental tension of PCGML with discriminative learning," in *Proceedings of the 14th International Conference on the Foundations of Digital Games*, ser. FDG '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3337722.3341845

[13] ——, "WaveFunctionCollapse is constraint solving in the wild," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, ser. FDG '17. New York, NY, USA: ACM, 2017, pp. 68:1–68:10. [Online]. Available: http://doi.acm.org/10.1145/3102071.3110566

[14] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural modeling of buildings," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 614–623, Jul. 2006. [Online]. Available: http://doi.acm.org/10.1145/1141911.1141931

[15] P. Filipe Coutinho Cabral D'Oliveira Quaresma, "A detail shape grammar. using alberti's column system rules to evaluate the longitudinal elevation of the nave of sant'andrea church generation," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 32, pp. 1–13, 04 2018.

[16] P. Merrell, E. Schkufza, and V. Koltun, "Computer-generated residential building layouts," *ACM Trans. Graph.*, vol. 29, no. 6, pp. 181:1–181:12, Dec. 2010. [Online]. Available: http://doi.acm.org/10.1145/1882261.1866203

[17] S. Lienhard, C. Lau, P. Müller, P. Wonka, and M. Pauly, "Design transformations for rule-based procedural modeling," *Comput. Graph. Forum*, vol. 36, no. 2, pp. 39–48, May 2017. [Online]. Available: https://doi.org/10.1111/cgf.13105

[18] H. Hua, "A bi-directional procedural model for architectural design," *Computer Graphics Forum*, vol. 36, no. 8, pp. 219–231, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13074

[19] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010.

[20] E. Teng and R. Bidarra, "A semantic approach to patch-based procedural generation of urban road networks," in *Proceedings of PCG 2017 - Workshop on Procedural Content Generation for Games, co-located with the Twelfth International Conference on the Foundations of Digital Games*, 2017. [Online]. Available: http://graphics.tudelft.nl/Publications-new/2017/TB17

[21] S. Snodgrass and S. Ontañón, "Learning to generate video game maps using markov models," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 4, pp. 410–422, Dec 2017.

[22] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider, "Potassco: The Potsdam answer set solving collection," *AI Commun.*, vol. 24, no. 2, p. 107–124, Apr. 2011.

[23] P. C. Merrell, "Model synthesis," Ph.D. dissertation, University of North Carolina at Chapel Hill, 2009.

[24] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Multi-shot ASP solving with clingo," *CoRR*, vol. abs/1705.09811, 2017. [Online]. Available: http://arxiv.org/abs/1705.09811

[25] M. Persson, "Minecraft," 2011.

[26] C. Salge, M. C. Green, R. Canaan, F. Skwarski, R. Fritsch, A. Brightmoore, S. Ye, C. Cao, and J. Togelius, "The AI settlement generation challenge in minecraft," *KI - Künstliche Intelligenz*, vol. 34, 2020. [Online]. Available: https://doi.org/10.1007/s13218-020-00635-0