Parametrics in Mass Customization

Charles C Vincent
Universidade Presbiteriana Mackenzie, Brazil
✉ cvin@arquitetos.com

Eduardo Sampaio Nardelli
Universidade Presbiteriana Mackenzie, Brazil
✉ enardelli@uol.com.br

Lia Raquel Nardin
Universidade Presbiteriana Mackenzie, Brazil
✉ lia_nardin@yahoo.com.br

ABSTRACT

The imminent disruption of the modern paradigms of serialization, repetition and standardization, poses us a myriad of new questions regarding both the emerging aesthetics and the upcoming production means for a new architecture. Despite the canonic approach in which architects tend to invest, the public and private demand for mass housing production is increasing at an astonishing rate, requiring architects to rethink traditional— modern—strategies and to gain control over contemporary—digital—tools. This paper describes an academic research project focused on the implementation of such tools. Some related work is presented, emphasizing some of the approaches in parametric plan layout generation. And a case study is formulated in which the mass design of serially customized layouts is prepared to be solved through a beta plugin for Rhino – Grasshopper.

KEYWORDS: mass customization, facade, digital architecture, aesthetics, production means.

Related Work

Many attempts have been made to generate plan and volumetric variations of a prototype design. During Negroponte's experiments the idea of embedding intelligence into a computation system was already being considered, and he suggested this could be accomplished in two ways: first, "directly embedding knowledge" in the form of facts and methods for manipulation of those facts, and second, "to impart in the machines the learning process itself" (Negroponte, 1975, 35).

For Terzidis the term "computation" should only be used if the machine process is one of processing information through a set of rules, may they be numerical or architectural ones. In this sense, he argues that "the codification of design intention using scripting languages" might as well help us depart "from 'architecture programming' to 'programming architecture.'" (Terzidis, 2006, 12)

Some works have described previous experiments with programming computation rule sets aiming specifically to generate variations in plan, facade or massing and that operate upon pattern generation, shape grammars, fuzzy logic modeling, evolutionary strategies, genetic algorithms, etc. In any case, "the plan is a choice from a set of plan-variants generated by the model. In a more complex design situation several (partial) models of the object to be designed generate, whether or not simultaneously, several sets of plan variants" (Bax & Trum, 2000). Another mechanism separates form from topology and translates space topologies into "spring forces" in a diagram generator (Arvin, 2002).

Yet another kind of experiment deals with user participation in the design process, such as Cheng's decision support system, which merges both user decision and layout generation scripts (Cheng & Lee, 2005).

Putting aside non-pragmatic evaluations based on cultural on non-contextual grounds, the problem of layout generation is difficult to solve whether by embedding "intelligence" into a layout generator by means of constraints (Shaviv, 1986, 31) or by testing layout solutions after generation.

Case Study

This experiment is currently under development; it was designed with a twofold objective.

First, it was devised as part of an educational approach in which undergraduate and graduate students were to be given the chance to formulate their own views on the mass housing issues—subjects such as variation in family profiles, mass customization, very low income customers, etc. Second, different programming strategies were discussed for problem formulation before script programming. The student group was split into two teams. The first team has been doing extensive research on mass housing solutions and industrialized construction and is developing a prototype unit design. The second group, made up of students with former programming experience and students trained in Rhino + Grasshopper, worked on specific scripting problems such as: list management strategies, grid manipulation and grid cell tagging, stochastic generators, conditional loops, etc.

Plan generation uses three strategies: 1) clients provide input for program requirements either 1a) graphically or 1b) tabulated, 2) programs are automatically generated, and 3) program requirements are input manually. Due to time constraints and the complexity of developing a graphical interface, the graphical input option was discarded early on, in favor of the tabulation option. Automatic generation was divided into suboptions: a genetic algorithm, shape grammar, and structured aggregation.

Two development methods are being researched: plan to elevation, where the plan distribution precedes the façade generation (Fig. 1), and plan to 3D, where a general 3D model is derived from plan distribution (Fig. 2).

Two programming or scripting tools are being considered at this stage: Grasshopper scripting and Excel programming to Grasshopper.

Many difficulties are encountered in Grasshopper scripting. Grasshopper is not capable of executing conditional statements and loops on its own. The generation process in Grasshopper thus involves the embedding of preset formal and relational rules, or constrains. Excel allows more sophisticated generation, since conditional loops can be employed to check if certain relations meet given criteria; moreover, the reference criteria may be non-linearly dependent of other results in the process.

In both cases, the team decided to establish the plan generation as a unidirectional procedure, because it was understood that a bidirectional process would entail almost insurmountable processing burdens. So, in the development of a typical plan, once one room is set, and once it is evaluated as correct it will no longer change; instead, the program will proceed to the next room, and so on.

In Grasshopper, a typical sequence will be:
- Check living room area
- Choose facade dimension (typically X) and calculate depth (typically Y)

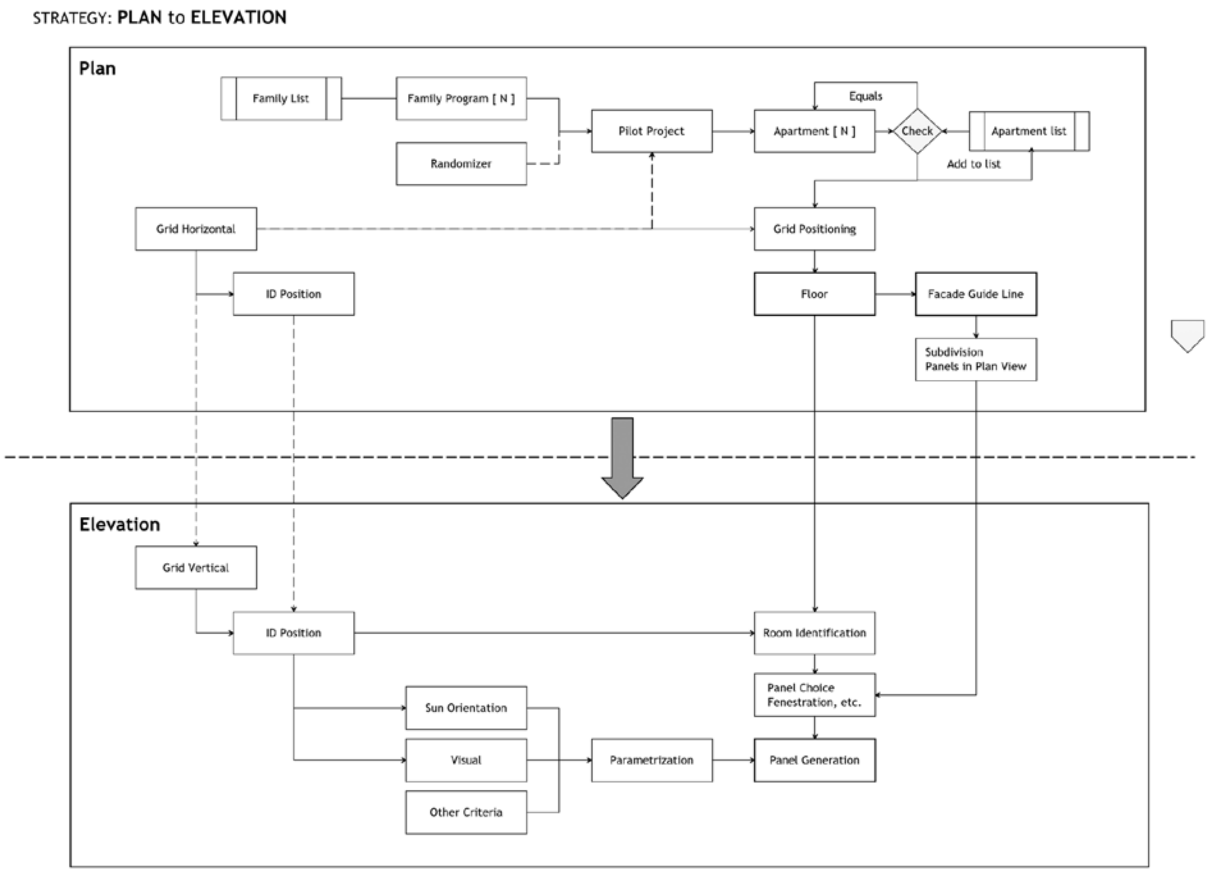

Figure 1. Plan to elevation development strategy

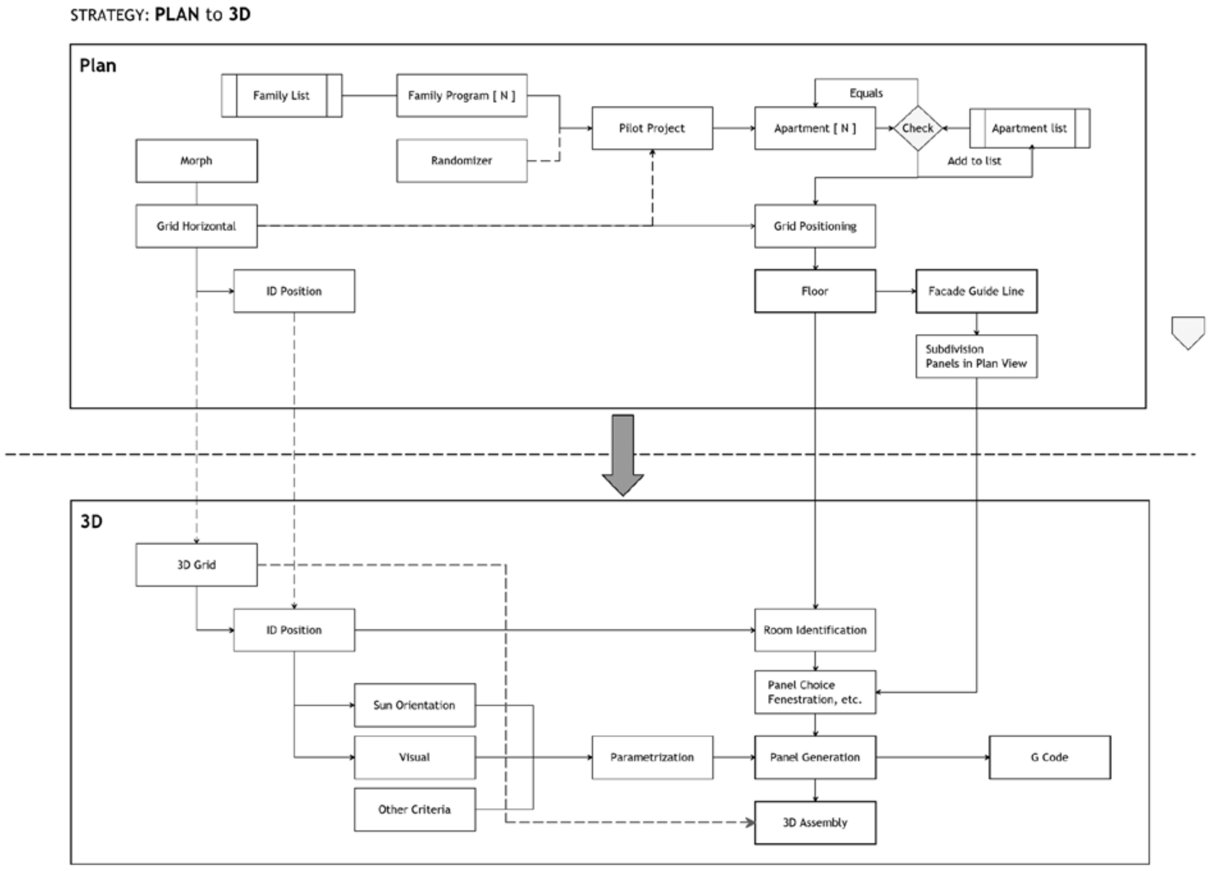

Figure 2. Plan to 3D development strategy

- Place living room
- Tag all occupied grid cells and store them in a list
- Check room area
- Choose façade dimension
- Choose next free cell and place room
- Etc.

Constraints to elements assigned in this manner can be set through transformations applied to the grid such as altering the grid according to rules such as numeric series or proportions or by morphing a base surface to which the grid is attached.

But the scripts established as described will lack constraints imposed between contiguous rooms, such as

If a living room is contiguous to a room, place a door.

In this scripting strategy, the façade or the 3D massing will be generated after the plan. As a result, the evaluation of the façade in terms of solar incidence, for example, will not affect the room dimensions, as should be desirable. However, upon façade analysis, each building fenestration will be evaluated on the outside in terms of solar incidence, sound insulation needs and, from the inside, in terms of which kind of fenestration is needed in individual areas (room, living room, kitchen, etc.).

The team faced other issues upon designing the plan generator, such as weather to freely generate layouts or to pick particular layouts from a list. The option of freely generating layouts implies that some testing mechanism should be applied after generation—for example, testing if proper access to all rooms was obtained, or if improper space contiguities exist. We decided that devising testing mechanisms would be immensely complex, and so it would be more effective to embed the “layout intelligence” into the plan generator.

A typical metacode is as follows:

- Pick a family program
- Check if a preset layout is present
- If preset layout is absent, randomly choose one
- Choose room 1 from the family program
- Apply preset rule and draw room 1
- Tag all occupied cells and add them to the cell list
- Choose room 2 from the family program
- Apply preset rule and draw room 2
- Tag all occupied cells and add them to the cell list
- Repeat this process as necessary

This is how a Grasshopper script behaves in the first tentative programs. In Excel another layer of complexity can be added, with some verification added after “placement” of all rooms is complete.

For instance, a discontinuity found in supposedly continuous lines can be corrected if a conditional loop is included after two rooms are "drawn". Such verification can be made to obtain facade or corridor alignment and, in some circumstances, both. However, the process is not directly linked to the graphic output in Grasshopper, instead, a generated list of coordinates and parameters that is fed into Grasshopper drives the geometry.

The issue remains one of deciding which constraints to embed and which ones to run after layout.

Conclusion

This is a project in development and, as such, very few results have yet been established. Besides having tested some preliminary scripts using both "embedded constraints" and cellular automata rules, the team still lacks further experimentation with topological diagrams for layout generation, which have already been experimented in projects such as Gen_House.

Some questions emerge. Can a generative plan script be developed that does not rely entirely on a prototype plan? How can formal and functional rules be reconciled?

Acknowledgements

This project is funded by the CNP, *Conselho Nacional de Pesquisa*, of Brazil.

Thanks to Bruno Leria, Victor Calixto, Victor Sardenberg, Julia de Andrade Reis, Leandro Gushiken and Tony Guedes, undergraduate students at UPM.

References

Arvin, S. & House, D. (2002) Modeling architectural design objectives in physically based space planning, *Automation in Construction 11* (2) pp. 213-225. Retrieved from: http://cumincad.scix.net/cgi-bin/works/Show?bcf7

Bax, T., & Trum, H. (2000) A Building Design Process Model According to Domain Theory, *Design Research in the Netherlands*. Retrieved from: http://www.designresearch.nl/PDF/DRN2000_Bax_Trum.pdf

Cheng, Yu-Chung, & Lee, Ji-Hyun (2005) Integrating scenario-based design and case- based design for user participation in Apartment plan design process, *CAADRIA 2005 - Proceedings of the 10th International Conference on Computer Aided Architectural Design Research in Asia*. New Delhi, India, April 2005, vol. 2, pp. 233-239. Retrieved from: http://cumincad.scix.net/cgi-bin/works/Show?caadria2005_b_4c_a

Koutamanis, A., (2001) Fuzzy Modeling of Floor Plan Layout, Reinventing the Discourse - How Digital Tools Help Bridge and Transform Research, *Education and Practice in Architecture: Proceedings of the Twenty First Annual Conference of the Association for Computer-Aided Design in Architecture*, Buffalo, NY, October 2001, pp. 314-321 Retrieved from: http://cumincad.scix.net/cgi-bin/works/Show?17ba

Negroponte, N. (1975) *Soft Architecture Machines*. Boston, MA: MIT.

Shaviv, E. (1995) Layout Design Problems: Systematic Approaches, *Computer-Aided Architectural Design Futures: CAAD Futures Conference Proceedings*, Delft, Holland, September 1985, pp. 28-52. Retrieved from: http://cumincad.scix.net/cgi- bin/works/Show?020d

Terzidis, K. (2008) *Algorithmic Architecture*, Oxford: Elsevier/Architectural Press.