Geometric Programming Framework

ANAR+: Geometry library for Processing

Guillaume Labelle¹, Julien Nembrini², Jeffrey Huang³
¹, ², ³ Media and Design Lab, EPFL, Switzerland
¹ (http://anar.ch)
¹ Guillaume.LaBelle@EPFL.ch, ² Julien.Nembrini@EPFL.ch, ³ Jeffrey.Huang@EPFL.ch

Abstract. This paper introduces a JAVA based library for parametric modeling through programming. From the recent advent of scripting tools integrated into commercial CAAD software and everyday design practice, the use of programming applied to an architectural design process becomes a necessary field of study. The ANAR+ library is a parametric geometry environment meant to be used as programming interface by designers. Form exploration strategies based on parametric variations depends on the internal logic description, a key role for form generation. In most commercial CAD software, geometric data structures are often predefined objects, thus constraining the form exploration, whereas digital architectural research and teaching are in need for an encompassing tool able to step beyond new software products limitations. We introduce key concepts of the library and show a use of the library within a form finding process driven by irradiance simulation.

Keywords. Processing; JAVA; Scene graph; Parametric Modeling; Geometry.

Architecture and Programming

Following the precedent of SketchPad, interactive drawing interfaces represent the initial penetration of computing in the architectural process. More recently, the introduction of parametric modeling techniques outlined a need for a better distinction between geometric representations and the logical structure of form. Despite the recent proliferation of programming interfaces for geometry, programming for design is yet a challenging field of study extending existing computer tools for design.

Programming as a creative process - Processing.org is a open source Java framework for designer and artists (Reas and Fry, 2007). At the time of its initial release, Processing proposed a different attitude toward programming, not anymore perceived as a purely technical matter, in advocating for a plurality of personal formalisms as an act of creation. In response to a software culture, where a pre-defined formalism is expected to be adopted by the user, Processing.org proposed a minimal set of elementary functions stimulating the user with the opportunity to define his own subjective formalism.

While the definition itself of a design problem is part of the creative process (Akin, 1986; Schoen, 1983; Simon, 1990) and depends intrinsically on specificities of project contexts. The designer, through his programming practice, defines, refines, redefines his own formalism, describing more
closely the nature of the design problem. Programming, instead of drawing, also matches the nature of the non-linear design process (Oxman, 2006) made of refinements where each step could compromise the project as a whole.

**Programming in Architecture**

Due to the geometric nature of architecture, formal thinking has always been part of the practice (Cache, 1995; Mitchell, 1990; Serres, 1993). Despite the recent proliferation of programming interfaces within popular CAAD software, only few examples introduced a structure intended to be primarily used through a programming interface. We outline here three different types of commonly found programming integration:

Integrated Scripting - Scripting refers to a programming language relying extensively on a specific platform (a CAD software in this case). The script is not independent of the targeted software and cannot be executed outside of the platform. They are often based on more generic programming languages with possible variations from the original language. Scripting languages are partially a transposition of a graphic interface into a programming equivalent. Examples of scripting languages: MEL (Perl), RhinoScript (VisualBasic), ArchiCAD’s GDL, AutoLISP.

Geometric Languages - Geometric Languages are mostly procedures describing how a resulting geometry is created. The geometric languages describe a geometric output through instructions, instead of end values as used in file formats. As an example, instructions to describe curves geometry use parameters interpreted by a procedure to create the geometrical element. File formats instead, by describing each coordinates of points, faces, etc, usually discretize curves into linear elements. The discretized objects don’t exist, they are created by the interpreter.

Examples of Geometric Languages: PovRAY (based on C language), VRML, LOGO.

Geometric Library - Geometric libraries are meant to be used independently from graphic-based specified interfaces. They are based on a mid-level programming languages such as C++, Java or LISP and provide functions and algorithms for geometry manipulation. They can be invoked within the user’s own code. Examples: OpenGL (primarily meant for rendering).

The work presented here focuses on providing a geometric library to be used either in conjunction or in the same attitude as the Processing.org project for architectural design. It provides basic functions to manipulate geometrical elements and aims at enabling the designer to establish his own specific formalism for digital architectural design.

---

*Figure 1*

Left: A parametric model of a tower. Right: Excerpt of the code generating the core of the tower.
ANAR+: Object Oriented Geometry

Initiated by interests in exploring automatic form generation algorithms such as genetic algorithms or L-systems, the ANAR+ geometric library presented here was developed with the underlying aim to provide ways to control the amount of parameters defining a form, e.g. the degrees of freedom of a form. The intent is to explicitly leave the designer choose which degrees of freedoms are to be automatically explored and which are defined in relationship to these free dimensions. It results in a formalization framework providing more space to formulate design intents to be further elaborated by an automated form exploration technique (form finding).

Use of the ANAR+ Library

This library has been confronted to various practical contexts: Within the Phototropic Architecture (LaBelle et al., 2008, Nembrini et al., 2009) project, the aim is to instantiate an integrated system combining form generation and simulation in a continuous automation, often described as a feedback loop. The ANAR+ library consists on a substantial part of a more general aim to better integrate form generation, automated exploration, simulation and fabrication in a compound model (Oxman, 2006).

The framework has also been used in teaching to introduce digital design thinking in general terms, away from specific software peculiarities. (Nembrini et al., 2009)

Behind the geometric representation, the logic of a geometric construction is expressed through a formalism. Many different geometric constructions could lead to a same representation. This is one problem of the limitation of software performing operation over the representation while not keeping the process of construction. The logic might have different meanings depending on the process context. In the ANAR+ library, a formalism describe and keep an history of all relationship between primitives.

Providing a representation of the logical process enable the possibility to develop specific operations on the logical representation itself. The modularity of the geometric library is intended to be used with growth algorithms (such as Genetic Programming or L-Systems (Prusinkiewicz and Lindenmayer, 1990)) and to study combinatorial permutations in complex logical structures.

Form-finding cycles

Computational approaches introduced the ability to automate parts of the drawing production. While the drawing could be reproduced and altered by computational automation, the drawing is no longer understood as something static, but as a process that generate variants. These computational methods of form-generation could be used to generate a set of derivative forms based on the same process, reducing the technical limitations for approaches based on selection. Often described as a feedback loop (Oxman, 2006) or morphogenesis (Hensel, 2004),
these approaches suggest the emergence of a form through an automated process of exploration of possible designs. The feed-back loop introduces the idea of a cycle through elementary operations:

- From generation
- Evaluation
- Exploration

These operations could be automated into what Oxman describe as a Compound Model, a systems that integrate generative processes with simulations as well as fabrication constraints.

**Form Generation**
With the automated generation of drawings provided by a parametrization, several variants of the same construction could be produced at will. All variations define a set – finite or infinite - of possible designs representing a solution space. This set is depending on the process generating the variations. Through the definition of a form generator, the designer take design decisions, bearing differences with a traditional design process in the fact that decision about such a form generator are made to preserve the coherence of possible designs; while the traditional approach is primarily pursuing one main potential design.

**Evaluation through simulations**
Obtained forms are evaluated through simulation that could predict the behavior of generated solutions. Building simulations approaches have greatly evolved over the past decade and are more accurate. The simulation provides quantifiable observations of an investigated solution. This includes geometrical analysis, structural analysis (FEM), light simulation, costs simulations, etc... Introduction of computational simulations in architectural design is technically problematic on various points;

- Simulations require an in-depth understanding of the simulated phenomenon for a calibration with corresponding observed physical phenomenon.
- Interpretation of results may require experts.
- Simulations may require specific geometrical informations beyond geometrical descriptions.
- Evaluation requires quantifiable information.

Limitations of simulations in architectural design remain a challenging problem. Often seen as a optimization problem over quantifiable data, it may be argued that the formulation of the right goals is as difficult as solving the problem itself. Since the discovery of the goals is project specific, the goals are hypothesis of good solution and should evolve over the design's process.

**Exploration**
Recent approaches in automated evaluation of solutions spaces have proposed for architectural design such as Genetical Algorithms, Particle Swarm Optimization, Neuronal Networks. These approaches assume the existence of a set of potential solution to be explored and are a subset of a general form of solution space exploration. Ultimately, by the exploration of every possible solution will lead to a set of acceptable solutions according to the explicit definition of designs intentions.

**Designing with Code**
Replacing interactive drawing (GUI) as commonly known in CAD software by writing code instructions represents an opportunity to study the pre-rationalization of geometrical constructions. While CAD software may be more permissive on some geometrical constructions to support “intuitive” usage, programming languages are less ambiguous, and thus require from the architect to precise his design intents. The use of ANAR+ is meant to consist in a stimulation for rationalization by providing a environment for “rational sketching”.

**Interacting with solution space as a design practice**
During the design process, a designer has to redefine and choose among a set of potential designs. In order to exclude incoherent, degenerated or
unbuildable possible solutions, the designer establish different strategies to exclude non convincing solutions. Again, these strategies clearly represent design decisions. Thus, we observe that the designer is still taking design decisions, but they are different in essence.

**On the use of self-organization in CAAD**

A parametric scripting interface enables the exploration of different flavor of self-organization processes in a specific architectural context with the aim to unveil complexities displayed by yet a very simple setup. By showing how the use of realistic simulation can be invoked to induce differentiation and communication among the components, the study explores the use of self-organization for architectural considerations.

The concepts and mechanisms unveiled by the study of self-organization in nature are very appealing to architectural designers. For instance, the complexity and functionality of nests built by social insects resonate with complexity of human interactions and activities which architecture endeavors to host. Current trends towards “organic” architecture and the quest for processes generating such organicity also sheds light on self-organization as a source of inspiration.

Directly inspired from (Bonabeau et al, 1999) the study relies on the following definition of
self-organization: “Self-organization is a phenomenon in which the internal organization of a system increases in complexity through the interactions of its components. Such phenomenon relies on the following ingredients: positive feedback, negative feedback, amplification of fluctuation and multiple interactions. It usually displays the following properties: spatio-temporal patterns, co-existence of multiple stable states and existence of bifurcations upon parameter change.”

**Difficult translation**

Such an interest lately fostered many examples invoking emergence as a design process to generate architectural form (Menges, 2004), including among others (Hemberg et al, 2007; Hensel & Menges, 2008). However, responses to the question on how to translate self-organizing processes in the architectural context can take several forms which have different implications on meaning and outcome. In the following, they are gathered in two different categories:

**Literal translation**

Direct translation of natural inspiring phenomenons into architecture. Studying self-organization processes may lead to an understanding of how organic forms are generated. For example, social insect nests generate organization of space resulting from an agent behavior. When applied to architecture, human agents may not behave like social insects, but the form is literally transposed into an architectural form. Such an approach tends to overlook the question of correspondence of the processes governing self-organization with the physicality of the system under consideration. The formal output governs the use of self-organization phenomenons and may lead to a decontextualized uses of the original self-organization principles. This goes in contradiction with the properties of examples such as social insect nests, which rely in the tight and dynamic interplay of behavioral rules, sensori-motor abilities and material constraints.

**Embodied translation**

Creation of self-organization through a system inherent to the architectural practice. In this approach, the understanding of biology’s behaviors is done by an in-depth analysis of the interactions happening and reinterpreting these in the context of the architectural project considered. No analogy with existing biologic systems is researched. General principles leading to self-organization are transposed into an “unnatural body” (Simon, 1969). This approach is related to the notion of material system as defined by (Hensel & Menges, 2008). It implies to integrate notions of performance and fabrication constraints in the process.

**Design-specific difficulties**

Beyond a organic formalism, the use of self-organization in design is however not straightforward. The analytical approach of biological studies of self-organization relies on a process inverse to the synthetic approach required by design.

**Rule design vs direct design**

First of all, to steer a self-organizing system toward a desired outcome can be highly difficult. The designer needs to delegate part of direct design operations to automatic processes and shift his attention on the design of rules steering such processes. Depending on the complexity of the system, the task to design and explore the outcomes of such rules can be counter-intuitive.

As a result, it may channel designers on the path of top-down approaches only mimicking self-organization (for instance by the fact that they use agents) or on choosing interpretations of self-organized outcomes strongly reducing complexity, thereby undermining the premise of using of self-organization altogether.
Danger of overcomplexity
Secondly, with plenty of algorithms/code available such as boids (Reynolds, 1987), ant colonies, particle systems, etc and the wider integration of programming interfaces in CAD software, it has become easy to set-up utterly complex systems. Thorough exploration of the dynamics of such systems could easily take years of research to master. In such cases, self-organization may appropriately be replaced by a crafted use of randomness.

Groundedness and embodiment
Emergent processes need to be grounded in a specific – possibly artificial – environment to interact with. Otherwise, the emergent properties of the system are likely to exploit non-realistic flaws of the system, e.g. simulation artifacts. The context of architecture calls to include in this modeled environment real physical simulation, such as structural or building physics simulations as used by civil engineers, thus providing strengthened confidence in the soundness of the results.

Analogous to the grounding problem, the notion of embodiment stresses that the simulation of emergence requires elements to have a body to interact with the environment. Self-organizing elements lacking embodiment may display behaviors breaching fundamental constraints of the environment considered. In such cases, the analogy between model and reality may be difficult to preserve, leading to a metaphorical use of self-organization. The question of fabrication and assembly directly relates this notion to the field of architecture.

Focus on key concepts
In the proposed paper, we will focus on concepts key to the use of self-organization in design contexts. We will explore and exemplify implications of different approaches through the help of an idealistic architectural case study. Relying on the definition of self-organization stated at the beginning, the set of such concepts will encompass the following dimensions, all having impact on the dynamics and outcomes of the self-organization process:

- **growth vs modification** Self-organization can happen in growing media as well as in stable population media. Results can be quite different in the two cases.

- **nature** of interactions Interactions can be direct or through the environment (stigmergy). Moreover, such interactions may convey semantic content or not.

- **feedback processes** The precise setup by which the state of components or the environment influence the future behavior of such components can take different forms.

The key concepts outlined here contribute to establish a taxonomy able to classify morphogenetic form-finding processes used in architectural design. Such a taxonomy supports critical thinking on algorithmic processes that are inherently difficult to grasp from traditional design communication medium such as project renderings.

Realistic simulations
As a possibility to establish a link between the use of emergent design and real-world constraints relevant to the architectural project, the coupling of form generation modeling with realistic building simulation provides self-organizing interactions with information on quantities such as energetic or structural behavior, thereby informing emergent processes with real-world constraints and contextual specificities.

References
Bonabeau, E Dorigo, M and Theraulaz, G 1999 ‘Swarm Intelligence - From Natural to Artificial Systems’, Oxford Univ. Press.


