ABSTRACT: 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 presented research explores the use of programming as explorative and reflexive medium (Schön, 1983) through the development of a programming framework for architectural design. Based on Java, the ANAR+ library is a parametric geometry environment meant to be used as programming interface by designers. Form exploration strategies based on parametric variations depend 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.

KEYWORDS: Parametric design, programming language, architectural Geometry, processing

RÉSUMÉ : Depuis les récentes avancées en matière d’intégration dans les logiciels commerciaux et les pratiques quotidiennes de langages de programmation, il est devenu nécessaire d’en faire un domaine d’étude. La présente recherche étudie l’usage de la programmation pour l’exploration et comme media réfléctif (Schön 1983) au travers le développement d’un cadre de programmation pour la conception architecturale. Basé sur Java, la librairie ANAR+ est un environnement paramétrique géométrique destiné à être utilisé comme une interface de programmation par des concepteurs. Les stratégies d’exploration formelle basées sur des variations paramétriques dépendent d’une description interne logique, un rôle clé pour la génération formelle. Dans la plupart des logiciels commerciaux de CAO, les structures de données géométriques sont souvent des objets prédéfinis, contraignant ainsi l’exploration formelle, alors que la recherche et l’enseignement de la conception architecturale ont besoin d’un outil englobant non assujetti aux limitations des nouveaux logiciels.

MOTS-CLÉS : Design paramétrique, langage de programmation, géométrie architecturale, processus
1. ARCHITECTURE AND PROGRAMMING

1.1. Automated Drafting

Integration of advances in CAAD research has typically been only slowly and gradually implemented within ordinary architectural practice, i.e. within commercial CAD software commonly used by architects. Several ideas originating from the “Robot Draftsman” project (RD aka SketchPAD (Sutherland 1963) ) can be seen as an important historical precedent (Mark 2008). This research challenges the nature of the fundamental changes introduced by automated drafting tools in architectural design. A second reading of RD underlines many key concepts for CAAD which are still sometimes presented as innovations and helps to understand the fundamental change from handmade technical drawings to automated process drawing.

- **Parametric design**: the use of enclosures (“sub pictures”), geometric associativity (“attachments”) and constraints—prefigures the definition of parametric design in which numerical variables are linked together to describe geometrical relationship between entities.

- **Continuity of scale**: the fluidity between scales allows for the coexistence in a single description of different levels of details that were traditionally developed in separate drawings.

- **Geometric variations**: automation of drawing production facilitates geometric variations towards a creation of a family of forms.

- **Simulation**: interfacing form definition with simulation provides the confrontation of design choices with approximations of reality.

- **Rationalisation**: the use of computers for design implies the disambiguation of form definition (Stiny 1993), which may precede or follow ideation. This is referred as pre- or post-rationalization, whose implications CAAD detractors have outlined.

At that time, RD emphasized the benefits of an graphical interactive tool for geometrical constructions. Today, most of current commercial CAD software can be seen as an extended development of RD concepts. The development of the ANAR+ library retains key programming concepts from this precursor, however decoupled from any graphical interface as a commonly adopted and rarely questioned feature of CAD, in order to study the internal logic of forms and the description of geometric relationship.

1.2. Code as Design Process

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 an 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; Schön 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.

### 1.3. 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.

2. 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 dimensions 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).

**FIGURE 1. REPRESENTATIONS; VISUAL AND LOGICAL.**

2.1. Logical and Visual representations

The study bases itself upon the observation that processes leading to geometrical results represent singular form thinking and should be expressed differently than a strictly descriptive model. Moreover, the design process embeds implicit design choices that often tend to be hidden in common CAAD software. The process of form definition is believed to be highly personal, revealing how the form is thought (Mitchell 1990; Stiny 1993).
In order to facilitate a variety of design processes to coexist, an open formalism is proposed built on well established elementary geometric concepts. Aiming to combine parametric (e.g. CATIA), dynamic (Cinderella) and procedural (LOGO) geometries, the ANAR+ library offers basic 3-dimensional environment and interaction, allowing designers to concentrate on form definition. Variations can be easily explored through the use of built-in graphical interface, in the form of sliders, controlling implicitly extracted parameters. Based on the combination of transformations of elementary geometric entities, the model better describes the constructional process and design intents which represent a particular and subjective perspective on an evolving problem context.

Built on a similar programming for non programmers attitude as the open-source framework Processing.org (Reas and Fry 2007), the formalism introduces careful simplifications to ease form definition, while allowing the invocation of extended components of a full-fledged object-oriented programming language. However easily used in independence as a geometric library, the framework is developed as an extension for the Processing.org environment (IDE). It provides a minimal framework (camera, texturing, OpenGL functions, etc.) for testing, learning and sketching. Conceived independently to a graphic interface, the library is autonomous and does not rely on external libraries. The geometrical description is based on linear algebra, well documented and widely used in computer graphics and CAD software. In order to allow the use of the created design in usual architectural practice, the library includes exporters to several CAAD scripting languages and popular geometrical file formats.

As a direct effect of the introduction of computers in the process of architectural design—primarily to help with technical representations and drawings—no clear distinction have been made between the representational model and the logic behind a representation. The study presented here aims at exploring the design potential of strongly decoupling description from representation. This conceptual aim is achieved through the use of coding alone to formally describe the form. If a graphical representation is used to evaluate the outcome of the formal description, modifications in the structure of the geometrical construction may only happen through code modification. It must be emphasized here that the presented research does not advocate for an exclusive use of programming in architectural design but for the necessity to explore constructional formalisms for their specific benefits to the design process.

The following sections elaborate on the topological structures and the parametric implementation mechanism implemented in the ANAR+ library.

2.2. Primitives

The following section outlines geometric entities provided as primitives. Using the JAVA inheritance structure, primitives could easily be extended to create
more specific elements while providing a common modular structure for similar objects.

**Parametric Framework:** To create dynamic geometric structures, every geometric instance needs to share a common mechanism where a single modification to an element propagates through all dependent objects. Also called scene graph, this mechanism remains simple and is applied indiscriminately to all objects of the library. As minimal mechanism, each modification to an entity triggers the rebuild of the objects that are based on it (children). Each entities have their own build methods with local values and could be rebuilt on demand according to different sets of initial values.

Remaining minimal, this solution ensures a uniform way to interact with objects while preserving the overall parametric coherence.

**Constructional Process:** The expression of a variety of form thinking is kept inside the logic formulation of the form. Every decision is embedded in the parametric framework where original instances are kept intact. The history of this process can be refined afterward. The description done through enclosures of finite sets of local condition provides an independence between instances. This is helpful for reusing or concurrent process.

**Numbers - Design with Ratios:** In ANAR+, distinct numbers are conceived as parameters of a form. Relationship among parameters ensure to keep the amount of parameters as low as possible. Replacing measurements by a set of simple mathematical relationship is the basic associativity mechanism. This mechanism can be reproduced when parent parameters are updated and parameters are chained from one to the other, recursively. The addition of simple relationship based on local conditions creates complex structures that can in turn express more complex relationship. This is done in order to reduce the use of static geometric descriptions while providing a way to keep intact the process of geometrical construction. The substitution of numbers by ratios improves the flexibility of geometric constructions enabling conditions to be easily be updated.

For example (Figure 2), a square could be expressed using only one value (the length of one side). If the same length is reused through the construction, the properties of a square are kept. A rectangle can be described with two length for each side, but it could also be described with two values: the ratio of its sides and the scaling factor. Changes to the parameters, i.e. changing the original values, will create a different form where geometric characteristics are kept. Implicitly, through the formalization of an object, all parameters are extracted, providing different ways to interact with the form. This structure creates an enclosure where the description of local conditions influences the resulting form.

**Interaction with parameters:** Parameters provide a generalized way to describe numerical values. The influence of parameters can be interactively
explored by the manual modification of values. On a simplest form, parameters are represented as graphical sliders through a built-in process. As every parameter provide a similar structure, this structure can easily be adapted to different kind of physical interfaces or can be controlled by an automated process.

**Absolute Points**: Points combine n parameters (representing each coordinates values) according to the dimension of a point. 2 parameters is considered as 2D point, 3 parameters is considered as 3D, ... nD. A point could contain the same parameter twice resulting in a constrained point on x=y slope.

**Derived points**: Derived points are depending on their parents, contrary to absolute points, they cannot be modified directly as they are the result of a relationship. They enable the expression of relationship such as points in the middle of two points, intersection between two lines or three planes, center of a face, transformed point (see Transform primitives), points constrained on a line.

**Points List and Faces**: lines (segments), multilines (polylines) and faces share a common data structure. The structure relies on a list of points. Depending on the object, they have different behaviors. The major distinction between face and multilines is in their displaying procedures: faces will act as closed filled polylines, by assuming that the user is careful in ensuring their planarity. Non-planar cases are only handled by the display method.

**Object**: is a combination of faces, points and lines. In ANAR+, while the primary focus is a logic representation, the Object doesn’t have prescriptive structures such as BREP data structures with normals orientations. An object could be composed of a point 1D or line 2D.
Transformations: are often expressed in the form of push and pop matrix stacking. While it provides an efficient way to combine matrices, it tends to make it more difficult to evaluate the actual position of a vertex. This is due to the OpenGL optimized calculations where real-time renderings are crucial. One of the drawbacks of this approach is that the evaluation of points coordinates is only created for a short period, at the time of rendering. This is problematic for procedures relying on coordinates such as area calculations, distance between points. In CAD systems, on the other hand, point coordinates are kept into a database and do not represent the process of generation. Such a geometry without generative process is denoted as an orphaned geometry, in which the geometry cannot be transformed by updating process parameters. ANAR+ library combines both strategies by creating intermediary geometries while keeping the constructional process. This provides a mechanism closer to the traditional CAD structure. Each construction steps result into a geometry that can be reused in a further process only. The proposed approach combining generation process and homogeneous geometrical structures inside a single framework is done by a generalization of transformation operators in such a way that each needed steps through the creation of an object are kept. The combination of transformations is the geoconstructional process.

2.3. Geometrical Relationship

The following section explains the formalism in ANAR+ describing the relationship between geometrical primitives. The primitives as inputs (parents) are combined together to create a new derived primitive as output (children). In programming terms, using object-oriented structure, the object have internal procedures describing how a geometric result could be obtained derived from the given inputs. This programming structure reduces a geometrical problem into a finite set of elements and can be reprocessed when the conditions change resulting in a propagation behavior. Resulting objects are themselves primitives and could be reused as parent for a new object, creating more complex structures. This simple recursive structure is enough to describe a parametric geometry.

A simple example is a point (M) between two points (A, B). The analytical relationship is the distance between the points (A, B) divided by two. (M) is the result of (A, B), if (A) or (B) is updated, (M) should reflect the new conditions. (M) could be himself used to calculate (N), the middle point between (A, M).

Simple shape: The next examples detail the representations of the cases presented in (Figure 2) using different geometrical constructions. A simple square is used as a point of comparison between different geometrical construc-

tions for the same representation. As outlined by (Stiny 1993), even a minimal shape is subject to many logical interpretations.

In (Figure 3), the shape is described by the 2D coordinates of the square vertices. The resulting shape family consists in deformed polygons, with regular polygons as special cases. The shape is composed of four points without relations. As seen, this geometrical construction is referred as an orphan geometrical description and is similar to a drawing without geometrical relationship, except to a referential coordinate system. If one point is moved, the other points remains at the same position. Found in early CAD software and for formats still commonly used today in the architectural practice (DXF, OBJ), this structure describes a geometry without any internal relationship between elements.

**FIGURE 3. GEOMETRICAL CONSTRUCTION OF THE SQUARE 1 BASED ON ABSOLUTE COORDINATES REFERENCES.**

*THE CONSTRUCTION HAVE 8 PARAMETERS REPRESENTING XY COORDINATES OF THE FORM.*

![Geometrical construction of the square 1](image)

The next construction (Figure 4) introduce transformations. The resulting family of shape includes rectangles, squares or sheared rectangles. In this case, the points are derived from a first point with a translation. The translation describes the relationship between two points on one edge. The first translation has only one parameter which constrains the point movement to one axis. The second transformation has two parameters which describe the potential movement of the second two points to the XY plane. In this construction, interaction with the system is done through the change of one point (the origin of the shape) or through the lengths of each side (translation in X or Y). A shear behavior is achieved with the 3rd parameter on the second translation.
Figure 4. Geometrical construction of the square 2 based on translations. The construction have 5 parameters: 2 for coordinates and 3 for the transformations.

The third construction (Figure 5 and Figure 6) is based on two different types of transformation in the same construction; translation and rotation. The resulting family of shape obtained is equilateral triangle, square, regular polygons (Figure 6), circle (with an infinity of edges).

Figure 5. Geometrical construction of the square 2 based on rotations and translations. The construction have 3 parameters.
The previous examples show how a different geoconstructional process leads to different meanings and behaviors for each construction. In fact, the list of different combinations could be extended infinitely. The formalism based on the predefined primitives allows comparisons to be drawn and outlines the logical structure of the construction of the geometry.

2.4. Architectural examples

Since the parametric relationships have a tree topology in graph-theoretic terms, abstracting this information and graphically represent it using the graph visualization algorithm `twopi` (part of the `graphviz` graph layout suite (Ellson et al. 2001)) enables the representation of the modeling strategies used in moderately complex examples.

**FIGURE 6. SIMILAR CONSTRUCTION AS PREVIOUS FIG. WITH A SMALLER ANGLE AND MORE EDGES.**

**FIGURE 7. RULED SURFACE CONSTRUCTION EXAMPLE. THE SURFACE IS OBTAINED THROUGH A ROTATION OF A NON PARALLEL LINE AROUND AN AXIS. RIGHT, CORRESPONDING GEOMETRIC CONSTRUCTION.**
FIGURE 8. GEOMETRICAL CONSTRUCTION FOR A CONCAVE FORM BASED ON TRANSLATIONS IN DIFFERENT DIRECTIONS.

FIGURE 9. ARCHITECTURAL EXAMPLE OF A COMPLEX GEOMETRIC CONSTRUCTION.
FIGURE 10. CODE EXCERPT FROM THE EXAMPLE TOWER.

```java
Face star = new Face();
int numberOfSides = 5;

Param w1 = new Param(50,0,100);
Param w2 = new Param(50,0,100);
Transform rz = new RotateZ(PI/numberOfSides);

Pt a = Pt.create(0,0,0).translateX(w1);
Pt b = a.copy().translateX(w2).apply(rz);

for(int i=0; i<5; i++)
{
  a = a.copy().apply(rz).apply(rz);
  b = b.copy().apply(rz).apply(rz);
  star.add(a,b);
}

Param height = new Param(5,0,30);

Param d = new Param(100,80,110).tag("%");
Param sx = new Param(95,90,110).div(d);
Param sy = new Param(94,90,110).div(d);
Param sz = new Param(104,90,110).div(d);
Param r2 = new Param(6,-20,20).div(d);
Transform complex = new Transform();
complex.translateZ(height);
complex.scale(sx,sy,sz);
complex.rotateZ(r2);

for(int i=0; i<24; i++)
{
  Face floor = star.copy().apply(complex);
  Anar.add(floor);
  star = floor;
}
```

In (Figure 10), a more complex formal construction represents the architectural skeleton of a building depicted in (Figure 9). The programming sequence remains simple. From this basic construction is derived the structural elements and inside walls on every second floor. The corresponding graph of internal parametric relationship displays a complex structure where the number of parameters is kept to only a few different values.

2.5. 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)

3. CONCLUSION AND OUTLOOK

Behind the geometric representation, the logic of a geometric construction is expressed through a minimal formalism. While a representation could share many different geometric constructions, the logic might have different meanings depending on the process context. In the ANAR+ library, a formalism is set to describe and keep an history of logical structures.

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. Also, the restructuring of the geometrical construction could provide promising avenues for parameters reduction, allowing for instance to constrain the search space for optimization techniques such as genetic algorithms.

ACKNOWLEDGEMENTS

This research has been supported by the Swiss National Science Foundation, KID grant number K-12K1-118078 and by the SiteMapping media art program of ProHelvetia (Federal Office for Culture), Switzerland.

REFERENCES