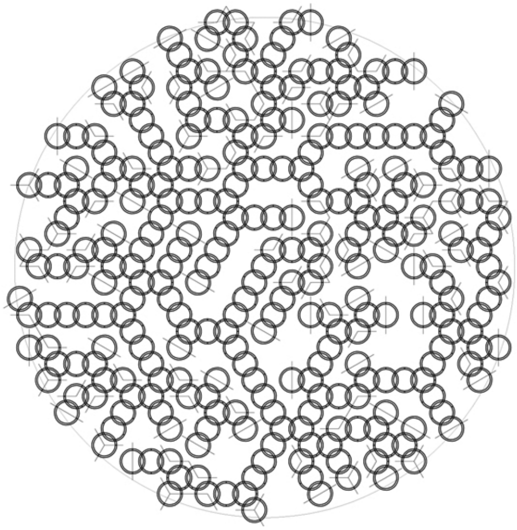


Figure 1. A design by Melnikov Grammar

The Melnikov Grammar

Daniel Cardoso, dcardoso@mit.edu, Massachusetts Institute of Technology
Takehiko Nagakura, takehiko@mit.edu, Massachusetts Institute of Technology
77 Mass. Ave of. 9-266, Cambridge MA 02139. USA

Abstract

The Melnikov Grammar is presented. A computer program that playfully re-interprets an iconic work of architecture by Konstantin Melnikov, implementing a rule-based system that semi-autonomously computes unexpected "Melnikov" designs in a non-deterministic manner while satisfying certain architectural constraints.

Keywords

Rule-based, Generative, Architecture, Melnikov, Shape Grammars

Overview

This study emerges from the authors' fascination with both the intriguing geometries of Konstantin Melnikov's designs and the computational nature of his design process, in which additions, subtractions and repetitions of a basic shape in accordance to a set of transformation rules configure a well-defined and open-ended formal language. By extracting the geometric rules of Melnikov's architecture, and defining a set of primitives for a language, the computational aspects of Melnikov's design process are encoded in a program (Melnikov Grammar), allowing for complexity and non-deterministic "Melnikov" designs. The program implements the rules to generate designs in a semi-autonomous manner, and satisfying certain user-defined constraints. The constraints implemented so far include a) a site's boundary, b) collision avoidance, c) a series of pre-existences, and d) the adaptation of the elements to maximize the views.

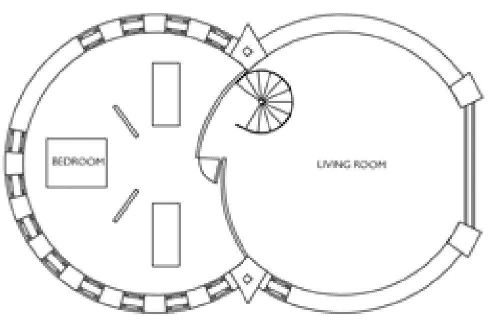


Figure 2. Melnikov House.

Background

Among the architects and artists of the second decade of the twentieth century in Russia, Konstantin Melnikov (1890-1974) stands prominently as one of its most intriguing an exceptional characters. His works as an architect can not be easily labeled as typical examples of the avant-garde of the period, nor can be fully explained by the formal principles of Russian constructivism. A closer look to the elegant patterns visible in some of his designs reveals instead the existence of concrete and precise geometric principles at play in his design methodology. Some key instances of Melnikov's architecture can be understood in this light, in their own terms, as having a particular generative logic. In other words, as a consistent exploration of a set of simple formal and geometric –as well as constructive- themes.

Perhaps the most important of Melnikov's built works is his own house (completed in 1929), a three storey construction consisting of the intersection of two cylindrical towers decorated with a pattern of hexagonal windows. The iconic presence of the house is very strong in part due to the platonic purity of its volumes, reminiscent of utopian projects such as Boullee's Newton Cenotaph, or Ledoux's Quarters for the rural caretakers [1]. The spiral staircase that acts as a connector between the different levels is located exactly at the intersection line between the two cylinders. The division walls are almost invariably a result of either radial lines (in the rooms) or of the projection of the missing wall of the cylinder (public areas). Other feature is that one of the cylinders has a flattened surface for a window, creating an asymmetry that differentiates the public tower from the private one, and affording the interior a different quality of light.

The cilyndrical form is a recurrent theme in Melnikov's architecture since 1924; the designs for the headquarters of the Leningrad Pravda in Moscow, and the Archos company are examples of this persistent trait. However, the generative principle that resulted in the Melnikov house was set in motion in the competition scheme for the Zuev Club (a sequence of 5 cylinders). In subsequent designs, Melnikov extended his geometric language of cylinders in his 1929 mass housing competition schemes, in which the cylinders were added not only in sequence, but also in clusters of five towers to form "three-petalled" plans that would later derive in the Burevestnik Clum building [1]. Eventually, accusations of formalism drove Melnikov out of the profession and he ended his life as a portrait painter.

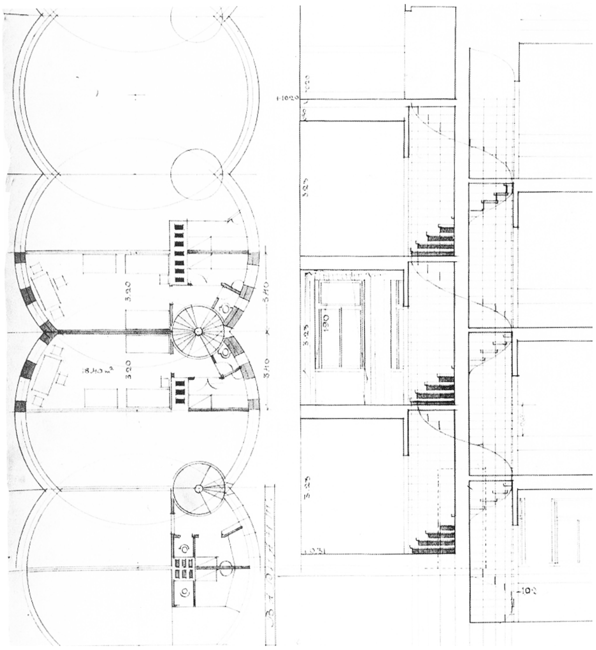


Figure 3. Mass Housing Competition, 1924

Rules

The elements and the rules are conceived as a shape grammar, a conceptual and logical framework developed by George Stiny, that provides a solid formalism for describing and generating designs through the specification of a set of ele-

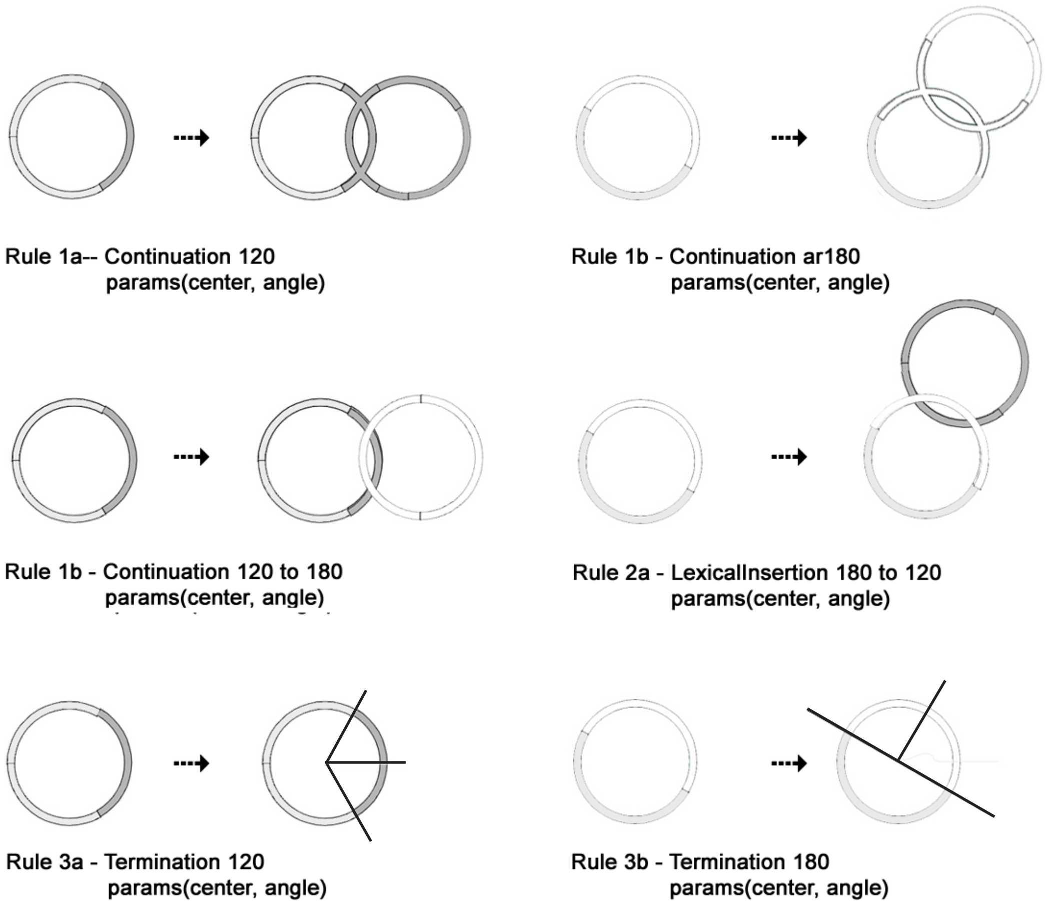

Rule 1a-- Continuation 120
params(center, angle)
Rule 1b - Continuation ar180
params(center, angle)
Rule 1b - Continuation 120 to 180
params(center, angle)
Rule 2a - LexicalInsertion 180 to 120
params(center, angle)
Rule 3a - Termination 120
params(center, angle)
Rule 3b - Termination 180
params(center, angle)

Figure 5. Rules of Transformation

ments and a set of rules for their transformation [2].

The grammar takes the basic design principles from Konstantin Melnikov's house and turns them into a set of explicit geometric rules. These rules are implemented in a computer program using the LISP programming language and NITROS, a prototype software developed by Takehiko Nagakura that provides a shell environment on which a description of formal elements and transformational relationships can be plugged in. NITROS imposes a rigorous framework to the describable relationships between the elements, which are typically described through parametric constraints; in other words, the value of some parameter of an element is computed from the values of the parameters of another element. Sorting out the constrained parameters and the free parameters is a key component of knowledge about a system of architectural types [3]. NITROS currently works as a plug-in for AutoCAD and is used mainly as a pedagogical tool.

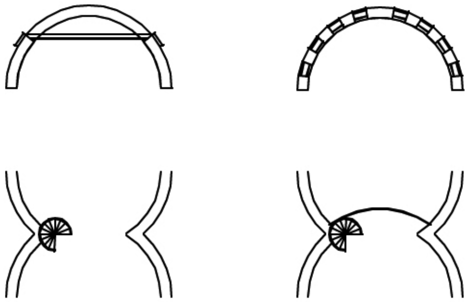
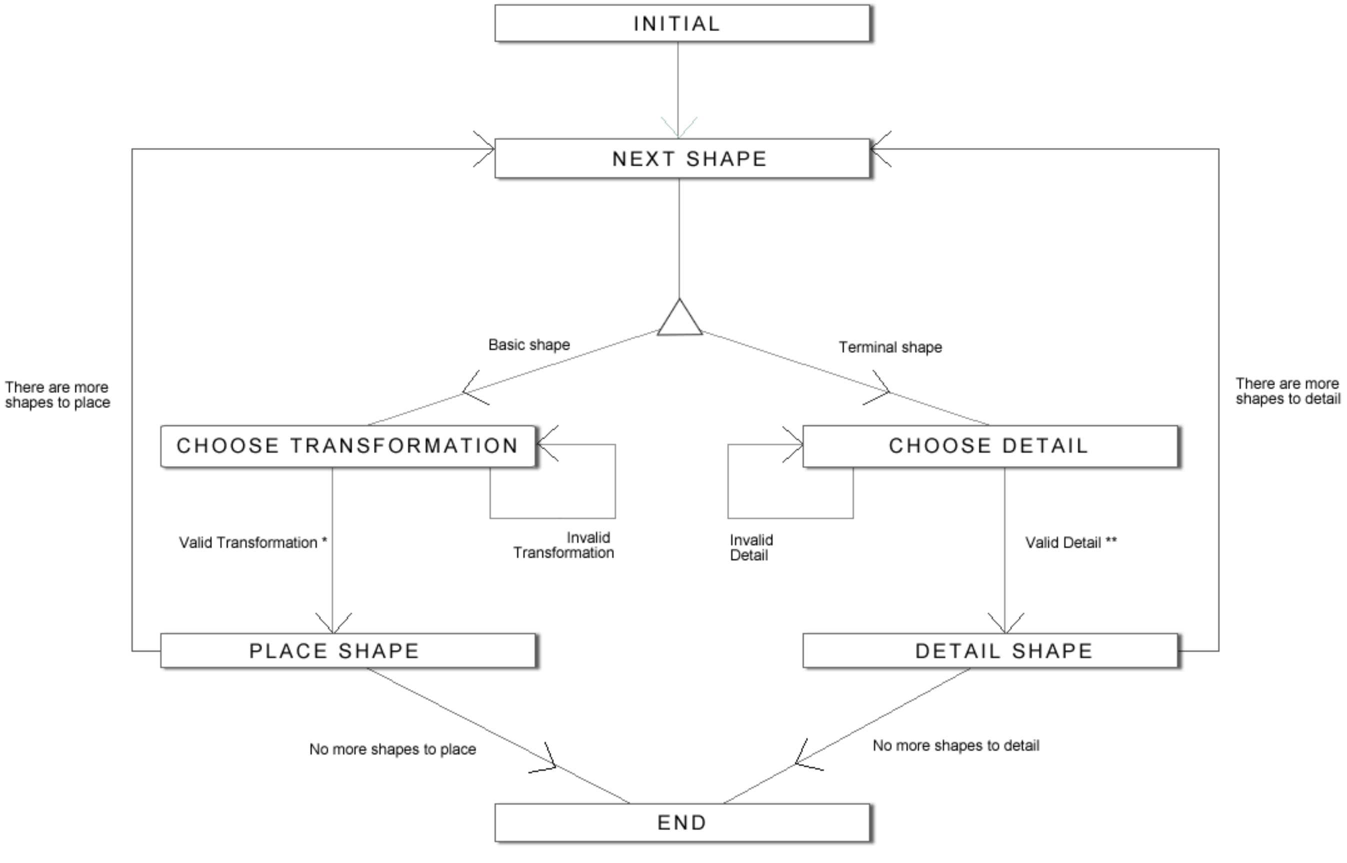


Figure 4. Some details to the primitive shapes

A first group of rules accounts for the connections between the primitive types, and therefore have morphological implications in the resulting 'urbanism' of the design. This group of rules essentially implements the two geometric principles of Melnikov's Architecture: a) the gram-



* The shape is within boundary and doesn't collide with other shapes

** The shape satisfies the view-finding constraint

Figure 6. Melnikov Grammar State Diagram

matical connections of cylinders divided in two sections (180 degrees), and b) the grammatical connections of cylinders divided in three sections (120 degrees). The geometric elements of this combination are defined by a scripted functions, and each one can be transformed and combined in a finite number of ways. These transformations typically substitute one element for another one that in turn can be transformed in a finite number of ways, allowing for emergent complexities and unpredictability despite the simplicity of the rules.

A second group of rules is in charge of detailing the shapes, adding windows, doors, and staircases. A detail rule can be applied only as a substitution of a basic shape that has no other transformation available (i.e. cannot be substituted by another basic shape), and are dependent on the view-finding constraint (see more details in next section), which means that a shape that is facing another shape at a very close distance is not likely to have a large window. Conversely, if the shape is facing an open space –like the site's boundary- a large window is likely to be placed.

On top of the combination logic of each component, the logic of the application of the rules determines what element to deploy in which circumstances; the following section describes in more detail the constraints that determine how the Melnikov Grammar is deployed.

Constraints

The application of a particular shape transformation depends on its satisfaction of a number of

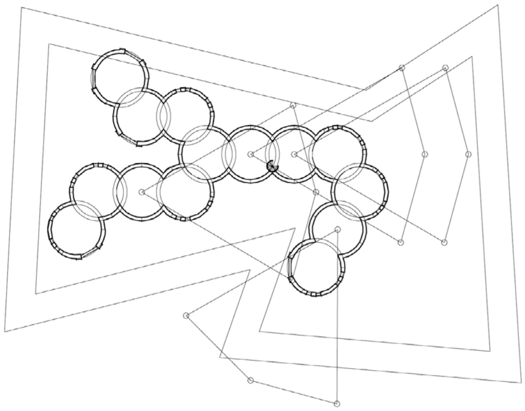


Figure 7. A derivation of the grammar constrained by a boundary.

constraints. The constraints built-in the system so-far are a) a collision constraint that is implemented as a collision detection algorithm which prevents a shape from being deployed –or substituted- on top of a previously deployed shape, b) an optional radius constraint that prevents shapes from being deployed outside a specified range, c) an obstacle constraint that detects and avoids a pre-specified object on site, d) a boundary constraint, that prevents the design to grow outside a pre-defined boundary, and e) a view constraint, that is meant to make sure that the kind of wall that the right kind of window or wall is placed at every element.

The following code accounts for the basic implementation of constraints a, c, and d.

```
(setq pop ( point-in-polygon newX newY world_
points_list ))
( setq flag 0 )
(setq allObjects( nt_visible_literal_geometries ))
( while allObjects
( setq this ( car allObjects ) )
( setq this_pList ( nt_plist this ) )
( setq thisType ( nt_type this ) )
( if ( = thisType “boxtacle” )
( progn
( setq thisX ( nt_val “xo” this_pList ) )
( setq thisY ( nt_val “yo” this_pList ) )
( setq thisZ ( nt_val “zo” this_pList ) )
(setq thisPoint ( list thisX thisY thisZ ) )
(if ( and
(and ( > newX ( - thisX 30 ) )
( < newX ( + thisX 30 ) ) )
( and ( > newY ( - thisY 30 ) )
( < newY ( + thisY 30 ) ) )
          ( = pop -1 ) )
          ( setq flag 1 ) ) )
( progn
( setq thisX ( nt_val “xo” this_pList ) )
( setq thisY ( nt_val “yo” this_pList ) )
( setq thisZ ( nt_val “zo” this_pList ) )
( setq thisPoint ( list thisX thisY thisZ ) )
( setq dis_current_to_new ( distance thisPoint
newPoint ) )
( if ( and ( = pop -1 ) ( < dis_current_to_new
15.1 )
( not ( and ( eq original-x thisX ) ( eq original-y
thisY) ) )
( progn ( setq flag 1 ) ) ) ) )
( setq allObjects ( cdr allObjects ) ) )
( if ( = pop 1 )
( setq flag 1 ))
( if ( = flag 0 )
t nil )
```

By enforcing these constraints while assigning transformations to a shape, the program is able to autonomously deploy architectural Melnikov shapes in 2d space. The state diagram in the pre-

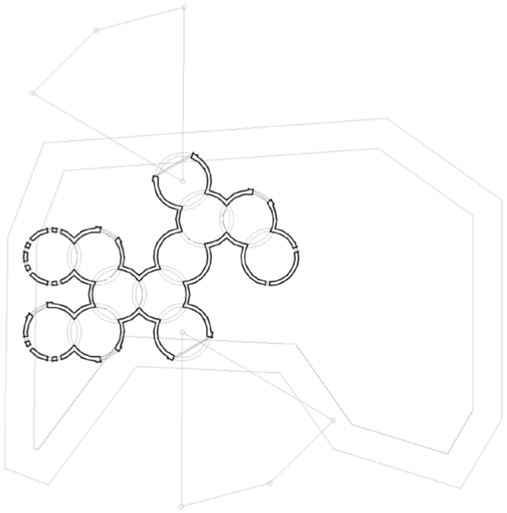


Figure 8. A plan view of a “smart” Melnikov Design

vious page describes the different states and transitions of Melnikov Grammar. Before applying a transformation, the program randomly decides which valid shape to transform and takes decisions based on the kind of shape chosen, and on the satisfaction of the corresponding constraints.

Results

By including collision detection algorithms and other constraints the Melnikov Grammar program is able to autonomously deploy architectural shapes in the virtual space, while following basic functional and lighting principles. Images in this page show some of the results of this autonomous process with different sets of rules and constraints. Work remains to be done in optimizing the collision detection algorithms, which cause the program to be very slow when a large number of shapes has been placed.

Summary

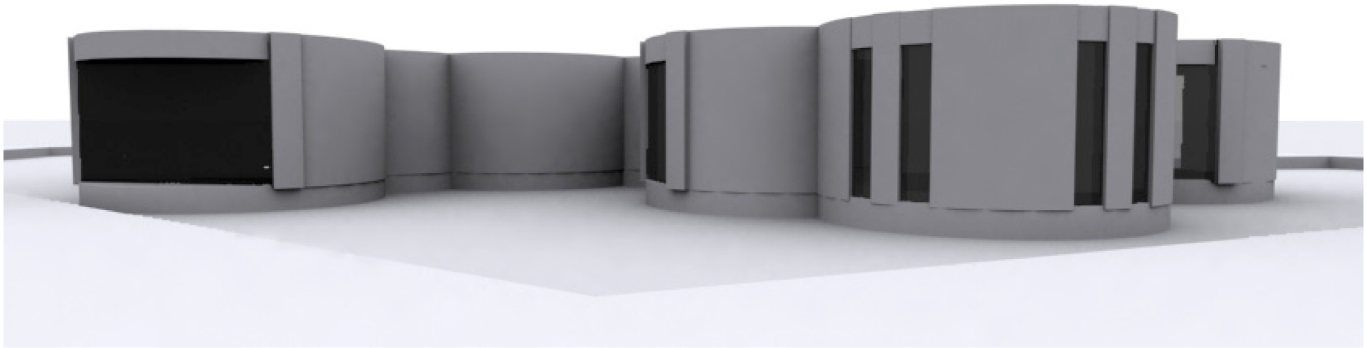


Figure 9. A 3-D rendering of a Melnikov Design

We have presented Melnikov Grammar, a playful computer program that implements a shape-grammar or rule-based system to generate architectural elements and urban morphologies non-deterministically while satisfying certain constraints like the boundary of a site, collision avoidance, and optimal view-finding. The elements that compose the architecture are a result of an exercise of appropriation and re-use inspired by Konstantin Melnikov's architecture, particularly by the Melnikov House built in Moscow (1929). Examples of the autonomous, or "smart" growth of Melnikov Designs by Melnikov Grammar, a detailed explanation of the computer program that implements the grammar, and renderings of the resulting morphologies are presented.

References

[1] Pallasmaa, Juhani and Andrei Gozak: 1996, The Melnikov House, Academy Editions, Singapore.

[2] Stiny, G and Gips, J: 1978: Algorithmic Aesthetics: Computer Models for Criticism and Design in Arts. University of California Press, Berkeley and Los Angeles, California.

[3] Nagakura, Takehiko 2006, Advanced Topics in Design and Computation: Implementing a small Top-down Grammar. Unpublished.

[4] Stiny, George: 1980, Kindergarten Grammars: designing with Froebel's building gifts. Environment and planning B 7(4): 409-462.