First International Conference on Design Computing and Cognition

Workshop 3

Implementation Issues in Generative Design Systems

Saturday 17 July 2004, 14:00 - 17:45
Table of Contents

Workshop goals and scientific committee

**Strict grammar-oriented approach**
A set-based shape grammar interpreter, with thoughts on emergence
*Andrew I-Kang Li and Lau Man Kuen, The Chinese University of Hong Kong, China*

Grammars, Sorts and Implementation
*Rudi Stouffs and Ramesh Krishnamurti, CMU, USA*

Shaper 2D: an interactive shape grammar interpreter
*Miranda McGill, MIT, USA*

*From simple to complex: using Autocad to build generative design systems*
*Gabriela Celani, UNICAMP, Brasil*

A studio exercise in rule-based composition
*Sotirios Kotsopoulos and Haldane Liew, MIT, USA*

Universal design + case driven computation
*Stylianos Drittas, MIT, USA*

**Other approaches**

Generative Design: from algorithms to intuitive and interactive software demonstrators
*Kristina Shea and Marina Gourtovaia, Cambridge University, UK*

Emergent Designer: An Integrated Research and Design Support Tool Based on Models of Complex Systems
*Rafal Kicinger, Tomasz Arciszewski, and Kenneth de Jong, George Mason University, USA*

Geometry as a substitute for structural analysis in generative design tools
*Martin Hemberg, Imperial College, UK and Una-May O'Reilly, MIT, USA*

Argenia, a mother tongue in infinite variations: identity rises from a soundless site, because birds have no tears
*Celestino Soddu and Enrica Colabella, Politecnico di Milano University, Italy*

Genometry: a genetically inspired parametric form generation method
*Anandasivam Krishnapillai, National University of Singapore, Singapore*
Hermes: a computational tool for proportionality analysis in design (15 + 5 min)
Athanassios Economou, Georgia Tech, USA

Poster Presentations

Down by the Law: a generative specification of US Court Houses
S. Dahabre and A. Economou, Georgia Tech, USA

Unfolding Geometries: a symmetry decomposition of Meier’s Smith House
E. Din, and A. Economou, Georgia Tech, USA
IMPLEMENTATION ISSUES IN GENERATIVE DESIGN SYSTEMS

1. Goals

Generative design is now an established research area. Several paradigms have been proposed such as shape grammars, genetic algorithms, parametric design, and so on. However, the impact on practice of any of these paradigms has been low so far. One can argue that one of the reasons behind such a low impact lies on the lack of computer implementations. This is somewhat surprising if one thinks that the computational basis of generative systems should naturally lead to the development of such implementations. The goal of the workshop is to understand the source and the nature of the difficulties that prevent those implementations from being developed. Papers describe existing implementations in their various aspects, and reflect on the theoretical and technical difficulties faced on their development. Issues such as shape representation, data exchanging, goal-driven strategies, interface and usability, platform compatibility, and paradigm deviation are addressed. Demos of implementations in various stages of development and depth (plug-ins, systems built from scratch, etc.) will be presented in the workshop.

Luisa Caldas and José Duarte, co-chairs

2. Scientific Committee

Kristina Shea, University of Cambridge, UK
Una-May O’Reilly, MIT, USA
Terry Knight, MIT, USA
George Stiny, MIT, USA
John Gero, University of Sidney
Chris Earl, Open University, UK
Andrew Li, Chinese University of Hong Kong, China
Hau Hing Chau, University of Leeds, UK
John Fraser, Hong Kong Polytechnic University, China
João Rocha, University of Minho, Portugal
A SET-BASED SHAPE GRAMMAR INTERPRETER, WITH THOUGHTS ON EMERGENCE

ANDREW I-KANG LI AND LAU MAN KUEN
The Chinese University of Hong Kong, China

Abstract. We present a set-based interpreter, implemented in Multimedia Flash, of a shape grammar for teaching the language of twelfth-century Chinese wood-frame building sections. We discuss the implementation of various aspects of the shape grammar formalism – shape representation, the part relation, and so on – the usefulness of the interpreter, and thoughts on the role of emergence in grammar interpreters.

1. Introduction

Shape grammars have a wide range of capabilities: emergence, parameterization, descriptions, labeling, weights, multiple drawings, and so on. However, interpreters that support all these capabilities have yet to be developed. Existing interpreters\(^1\) have all been restricted, in ways that derive from the intent of the grammars they implement.

Grammars intended as creative design tools (i.e., design or synthetic grammars) generally use emergence and matching under multiple transformations, but not extensive labeling or parameterization. Interpreters that support such grammars – let us call them synthetic interpreters – are restricted accordingly. Because of these restrictions, at least in part, Chase (2002, 162) observes that “[t]he user interactions in such tools tend to be rather limited in scope.”

Interpreters that support analytic grammars, or analytic interpreters, on the other hand, tend to be restricted in the opposite way: they use extensive labeling and parameterization, but not emergence or matching under multiple transformations. In this case, the restrictions seem less noticeable to the user.

Take as an example Flemming’s (1987) Queen Anne interpreter which, as he emphasizes, is based on a set representation (Stiny 1982) and so does

\(^1\) Gips (1999) provides a summary.
not support emergence. He does not need this capability; indeed, he “hardly
missed having a general shape grammar interpreter available” (Flemming
1987, 266).

In addition, he knows that the design space is restricted in another way:
“under the selected representation, properties such as the fact that certain
edges form a rectangle, are already implied and do not have to be laboriously
established whenever a rectangle is called for” (Flemming 1987, 268).

Thus the restrictions of analytic interpreters are not seen as shortcomings,
but those of synthetic interpreters are. Chase (2002, 162) suggests that the
shortcomings “may be due in part to … a lack of understanding how
grammars relate to the design process” and calls for “[f]urther research on
interactions in grammar systems.”

As a step in this direction, we present an analytic interpreter built for a
narrow purpose: to create wood-frame building sections according to the
twelfth-century Chinese building manual Yingzao fashi [Building standards],
by Li Jie (d. 1110). This interpreter is part of a larger scheme for teaching
the architectural style of this manual (Li 2003), a scheme which has led us to
specific ideas about the task for which the interpreter is a tool, the user’s
experience in completing the task, and the interpreter as a tool for the user.
These ideas have in turn led us to a specific conception of the interaction
between the user and the interpreter.

We find this interpreter to be largely satisfactory for its purpose, although
it also exhibits a telling shortcoming. By considering this example, we
derive some lessons about the relation between users and interpreters
generally.

2. Hypothesizing the language of sections

The Yingzao fashi is a prescriptive guide for building construction. Li’s
approach is in general not enumerative but generative, as noted by the
architectural historian Liang Sicheng (1984), who called it “a grammar
book.” By contrast, what Li has to say about building sections is not
generative but enumerative: a set of eighteen drawings and written
descriptions (see figure 1).

For us, a grammatical understanding of the language of which the corpus
is a part presupposes the following conceptual framework.\(^2\) The corpus of
sections is a set of empirical observations. The grammar is a hypothesis that
makes predictions (creates designs). The predictions are tested (the designs
are evaluated for stylistic correctness) and the hypothesis (the grammar) is
revised accordingly.

\(^2\) For a more thorough discussion, see Stiny and Mitchell (1978) and Li (2003;
forthcoming).
To teach within this framework, we devise the following scenario. We provide a grammar; the students test and revise it as necessary. We develop the grammar with the expectation that it should generate all and more than the sections in the language. Then the students’ task of evaluating and revising designs is to see and eliminate rather than to imagine and add. We expect further that students can eliminate designs by changing the constraints on schema application, not by changing the schemata themselves.

The grammar is a parametric set grammar with descriptions. Each initial design consists of a shape—a diagrammatic section of 6 rafters’ depth—and two descriptions (one Chinese, one English). The initial shape consists in turn of a ground line, two columns (front and back), purlin placeholders, a vertical axis line, labels, and a symbol. The initial descriptions are 6-jia chuan wu, ∅, yong 2 zhu, and 6-rafter building, ∅, with 2 columns (see figure 2).

The grammar has four stages. In the first stage (schemata A1–A21), building components, such as columns or beams, are inserted into both the section and the descriptions. In the second stage (schemata A22–A31), any remaining labels are cleaned up; the descriptions are left unchanged. In the third stage (schemata B1–B17), building components (beams, rafters, etc.) are inserted as necessary to complete the section. However, these components are not specified in the descriptions, which therefore remain unchanged. In the fourth stage (schemata B44–B48), the descriptions are reduced to a standard form.

These four stages are not equally relevant to the user, whose task, we recall, is to create and evaluate designs. Only the first stage is relevant, because it is there that the user chooses the features that distinguish the design within the language, that is, the salient features. It is nondeterministic and requires the participation of the user.

---

3 For more on descriptions, see Stiny (1981). For more on the algorithm, see Li (2001).
4 We use the term design following Stiny’s (1990, 97) definition: “an element in an n-ary relation among drawings, other kinds of descriptions, and correlative devices as needed.” In our case the design consists of a shape (the section) and two descriptions.
Figure 2. The opening screen of the interpreter. The initial shape and descriptions are above, in derivation view. The schemata (A1–A21) of stage 1 are below. Those schemata that can be applied to the initial shape are shown undimmed.

The other stages do not contribute any salient features. They either do grammatical housekeeping chores (stage 2), complete the section (stage 3), or complete the descriptions (stage 4). They are deterministic and of little interest to the user in the context we have described.

Thus we wish the user to pay attention to the schemata in stage 1 and to be relieved of involvement in stages 2, 3, and 4. To help make this happen is to us one of the main purposes of the interpreter.

3. The interface

3.1. REQUIREMENTS

Our situation contains the three entities of Chase’s (2002) model: developer (teacher), users (students\(^5\)), and interpreter. As already mentioned, we have developed the interpreter\(^6\) to create a specific pedagogical experience for the students; this is as follows.

\(^5\) We use students and users interchangeably.
\(^6\) An earlier version of the interpreter was reported in Li (2002).
The student makes design decisions (applies rules) to explore the language of designs. Thus the interpreter should help the student create designs easily and assimilate the algorithm. More generally, the interpreter should provide all the information that the user needs to make his decisions. In terms of the derivation of a design, it should tell the user where he has been, where he is, and where he can go. It should hide everything else.

In terms of revising the grammar, we expect that it should suffice for the user to articulate constraints verbally (e.g., do not apply the same schema twice in succession), and so do not provide for user modification. Given that in stage A the interpreter provides 367 schema sequences that create only 32 distinct sections, we expect that most modifications should involve filtering out inappropriate sequences, not adding schemata.

3.2. CONCEPTUALIZING THE INTERFACE

To discuss the interface, it is helpful to use some technical apparatus. Consider a design \( D = \langle C, d \rangle \), where \( C \) is the shape and \( d \) is the description. Recall that the next design \( \langle C_{i+1}, d_{i+1} \rangle \) is derived from the current design \( \langle C_i, d_i \rangle \) in the following way. For the shapes,

\[
\text{if } g(t(A)) \leq C_i, \text{ then } C_{i+1} = \lfloor C_i - g(t(A)) \rfloor + g(t(B)), \tag{1}
\]

where \( A \) and \( B \) are respectively the left and right shapes in the schema \( A \rightarrow B \), \( t \) is an appropriate transformation, and \( g \) is an appropriate parametric assignment. For the descriptions,

\[
d_{i+1} = f(d_i), \tag{2}
\]

where \( f \) is a description function associated with the schema \( A \rightarrow B \).

We propose that the interpreter should support the following scheme of user interaction.

1. The interpreter shows the current design \( \langle C_i, d_i \rangle \).

2. The interpreter shows which schemata can be applied to the current design \( \langle C_i, d_i \rangle \) under an appropriate transformation and parametric assignment. In other words, show those schemata \( A \rightarrow B \) with associated descriptions \( f \) for which \( g(t(A)) \leq C_i \) under appropriate values of \( t \) and \( g \). In stages 1 and 2, the search is simplified to \( g(A) \leq C_i \), because there is not more than one appropriate value of \( t \).

---

7 In the previous version, users did in fact have to execute stages 2 and 3 manually. (Stage 4 had not been implemented.) It was immediately obvious that this distracted users, and we added the capability of automatically deriving the deterministic sequences.

8 As José Pinto Duarte first pointed out, this interpreter actually implements \( g(t(A)) \) and \( g(t(B)) \), not Stiny’s formulation of \( t(g(A)) \) and \( t(g(B)) \). There seems to be no difference in outcome.
3 The interpreter shows the outcome $C_{i+1}$ of the schema application, where $C_{i+1} = [C_i - g(t(A))] + g(t(B))$.

4 The user chooses a schema.

5 The interpreter applies the schema and updates the current design. In addition, the interpreter should record the derivation $\langle C_0, d_0 \rangle, \langle C_1, d_1 \rangle, \ldots, \langle C_n, d_n \rangle$ and allow the user to undo schema applications back to the initial design $\langle C_0, d_0 \rangle$.

3.3. DESCRIPTION

The screen has two halves (see figure 2). The upper half is devoted to the current design. The user can toggle between two views: the current design and the next design in a large size (current transformation view; see figure 3), or the whole derivation in a small size in a scrollable window (derivation view; see figure 2). Both the Chinese and the English descriptions are shown. The derivation can be printed.

The lower half of the screen contains the schemata. These are divided over four “pages” corresponding to the stages mentioned above. Thus the first page, entitled "install named beam," contains schemata A1–A21; the second, "clean up labels," A22–A31; the third, "complete roof," B1–B17; and the fourth, "complete description," B44–B48. Only the shape schemata are shown; the description functions are shown in pop-up windows (see figure 3).

Schemata that cannot be applied to the current shape are dimmed. Those that can be applied are undimmed. The schemata in stages 1 and 2 can be applied under not more than one transformation and one assignment. Thus one state – undimmed – covers all possible applications.

In stage 3, by contrast, some schemata can be applied under two reflections and one assignment (see figures 4 and 5). Applicable schemata are shown under one of the two reflections; the user can toggle between them by clicking the mirror button. As a further help to the user, the schemata are also shown under reflection. In stage 4, there are no shape schemata, only description functions, so the question of transformations and assignments is moot.
Figure 3. Previewing the next design. The user has rolled over schema A2. The current and next designs are shown in current transformation view. The description functions are shown in a pop-up window. The elements to be changed in the shape and the descriptions are highlighted.

When the user rolls the mouse over an applicable schema $A \rightarrow B$, the next design is previewed in the upper half of the screen, and the associated description function appears in a pop-up window (see figure 3). Several objects are highlighted in red:

- The schema’s left and right shapes $A$ and $B$ (in stage 3, the left and right shapes under transformation $t(A)$ and $t(B)$, where $t$ is toggled);
- The corresponding subshapes $g(t(A))$ and $g(t(B))$ in the current and next shapes $C_i$ and $C_{i+1}$;
- Each description function $f_i$; and
- Those parts of the current and next descriptions $d_i$ and $d_{i+1}$ that are manipulated by the function $f$.

Thus the user can easily survey the possible next designs. He can then apply one of the applicable schemata or back up to the previous design and resurvey the possibilities.
Figure 4. The schemata of stage 3 shown under the first of two reflections.

Figure 5. The schemata of stage 3 shown under the second of two reflections.
When the user clicks on an applicable schema, he applies it under the available transformation and assignment. The next design becomes the current design, and the new set of applicable schemata are shown.

The user can generate a design by applying each schema manually from beginning to end but, as we have mentioned above, such close engagement in the deterministic stages is distracting. Thus in stages 2, 3, and 4, the user can click *complete automatically*, and the interpreter executes that stage of the derivation.

4. The implementation

The interpreter is implemented in Macromedia Flash, because Flash is relatively easy to use and allows extensive control over the interface. In addition, the projectors it creates are easy to distribute because they do not need player software.

We will discuss here, not the mechanics of the supporting capabilities—recording the derivation, undoing schema applications, etc.—but the implementation of the shape grammar formalism—shape representation, the part relation, and so on. As has already been mentioned, emergence is not required. Thus the interpreter uses a set representation.

In the discussion below, we follow the scheme of interaction given above. However, the last step, applying a schema and its description function, involves only housekeeping and no shape grammar representation; we omit it.

4.1. SHOWING THE CURRENT DESIGN

A design is represented as a table of values in the following way. It consists of a shape and two descriptions, and is a finite set of discrete elements. Each element has a finite number of attributes, and each attribute has a finite number of values. To display a particular design is simply to display its elements with the appropriate value of each attribute.

The shape is made up of elements that are either parts of the section (e.g., the ground line or a column) or control devices (labels and symbols). Each element has a fixed position and the following attributes:

- Intensity: hidden, dimmed, shown, or highlighted;
- Form: circle, square, single triangle, or double triangle (for labels); A, B, or C (symbols only); and
- Fill: hollow or solid (labels only).

The English description consists of text elements (letters, numerals, and other symbols). The Chinese description consists of images of Chinese characters, rather than characters as text elements, which Flash does not support.
Below is the code for the `Shape` function, which instantiates a design. Some parts of the section are represented as arrays, where the index indicates the part’s position between the front and the back of the building. For example, there are 7 possible column locations $i$, $0 \leq i \leq 6$ from front to back. A given column is represented as `column_var[i]`, where $i$ is its location. As we will see, this representation enables the parameterization.

```javascript
function Shape() {                     // creates both shape and
  // descriptions
  this.num_rafter = 6;                 // the depth of the building
  // other values are 4, 8,
  // and 10
  this.purlin_front = 0;               // locates the front purlin
  this.purlin_mid = this.num_rafter / 2; // locates the ridge purlin
  this.purlin_back = this.num_rafter;  // locates the back purlin
  this.purlin_var = new Array();
  this.column_var = new Array();
  this.beam_var = new Array();
  this.roof_var = new Array();
  this.control_var = new Array();
  this.controlclean_var = new Array();
  this.baseline_var = new Array();

  for (i = 0; i <= 6; i++) {           // 6 = this.num_rafter
    this.purlin_var[i] = "circ_hol_show";
    this.column_var[i] = "hide";
    this.control_var[i] = "circ_hol_show";
    this.controlclean_var[i] = "hide";
    this.baseline[i] = "show";
  }
  for (i = 0; i <= 5; i++) {           // 5 = this.num_rafter - 1
    this.beam_var[i] = "hide";
    this.roof_var[i] = "hide";
    this.vertaxis_var = "show";
    this.controlfront_var = "show";
    this.controlback_var = "show";
    this.stage_var = "A show";
    this.column_var[this.purlin_front] = "show";
    this.column_var[this.purlin_back] = "show";
    this.control1_var[this.purlin_front] = "tri_sol_show";
    this.control1_var[this.purlin_back] = "tri_hol_show";
    this.controlClean1_var[this.purlin_front] = "sq_sol_show";
    this.controlClean1_var[this.purlin_back] = "sq_hol_show";
    this.c = 2;
    this.be1 = "6-rafter building";
    this.be2 = "∅";
    this.be3 = "with " + String(this.c) + " columns";
    this.bc1 = "six_jia_chuan_wu";
    this.bc2 = "nil";
    this.bc3 = "yong_" + String(this.c) + "_zhu";
  }
  shape_current = new Shape();
  shape_new = new Shape();
}
```
4.2. CHECKING THE APPLICABILITY OF SCHEMA A2

Recall that the schemata that can be applied to the current design are shown undimmed. How the interpreter determines the applicability of each schema is best seen by looking at two sample schemata: one with a single transformation, a single assignment, and descriptions; and the other with two transformations, a single assignment, and no descriptions.

The first sample schema A2 inserts a one-rafter-long beam and a column into the current shape and the descriptions. It can be applied to a current shape under a maximum of one transformation and one assignment. Thus the interpreter does not need to determine the transformation; it has only to test the possible assignments. The code is shown below.

```java
if (check (_root.shape_current) <> -1) {
    rule_left = new _root.Shape();
    rule_right = new _root.Shape();
    init_rule (rule_left, rule_right, check (_root.shape_current));
    gotoAndPlay ("show");
}
stop();
```

There are two functions here: check and init_rule. The first, check, takes the current shape shape as its argument and returns the parametric assignment i under which schema A2 can be applied to shape; if there is no such assignment, it returns –1. It does this by examining each position i in the front half of the building, 0 ≤ i ≤ purlin_mid. For each value of i, it checks whether the value of each attribute of each element in the left shape of the schema matches the value of the corresponding attribute in the current shape, that is, whether g(A) ≤ Ci. In other words, the part relation ≤ is implemented as matching pairs of values; the transformation t is moot; and the assignment g is the index i of an array.

```java
function check (shape) {
    for (i = shape.purlin_front; i < shape.purlin_mid; i++) {
        if (shape.column_var[i] == "show" and
            shape.column_var[i + 1] == "hide" and
            shape.purlin_var[i] == "circ_hol_show" and
            shape.purlin_var[i + 1] == "circ_hol_show" and
            shape.control_var[i + 1] == "circ_hol_show" and
            shape.control_var[i] == "tri_sol_show") {
            return i;
        }
    }
    return -1;
}
```

The other function init_rule sets values to be used if the schema is applied. It takes as arguments the shapes left and right and the assignment t returned by check. It sets the value of each attribute of each element in the left shape of the schema to na; it sets all other elements to
none. This set of na values is in effect the “inverse” of the left shape of the
schema. As we will see below, it will be added, not subtracted, if the next
shape is calculated. In other words, init_rule prepares \(-g(A)\) (as it were)
and \(g(B)\) to be used if the next shape \(C_{i+1} = [C_i + (\text{-}g(A))] + g(B)\)
needs to be calculated.

```javascript
function init_rule (left, right, t) {
  for (i = left.purlin_front; i <= left.purlin_back; i++) {
    left.purlin_var[i] = "none";
    left.column_var[i] = "none";
    if (i < left.purlin_back) {
      left.beam_var[i] = "none";
      left.roof_var[i] = "none";
      left.control_var[i] = "none";
      left.controlClean_var[i] = "none";
      left.baseline_var[i] = "none";
    }
    left.vertaxis_var = "none";
    left.controlFront_var = "none";
    left.controlBack_var = "none";
    left.stage_var = "none";
  }
  left.purlin_var[t] = "na";
  left.purlin_var[t + 1] = "na";
  left.control_var[t] = "na";
  left.control_var[t + 1] = "na";
  left.stage_var = "na";
}
```

9 This approach offers the slight advantage of requiring only an addition function, rather
than both addition and subtraction functions. However, it is inconsistent with the formal
definition of rule application and so is not ideally clear.
right.control_var[t + 1] = "tri_sol_show";
right.stage_var = "A_show";
}

4.3. CHECKING THE APPLICABILITY OF SCHEMA B12

The second sample schema B12 differs from A2 in having two reflections under which it may be applied to the current shape. It does two checks, one on each reflection, as seen in the code below.

```javascript
if (/:mirror == 0) {
    gotoAndStop ("check");
    if (check (_root.shape_current) <> -1) { gotoAndPlay ("show"); }
} else {
    gotoAndStop ("check_reflect");
    if (check_reflect (_root.shape_current) <> -1) { gotoAndPlay ("show_reflect"); }
}
stop();
```

Check and check_reflect both work like check for A2 above. They return the assignment i under which the schema B12 can be applied to a shape (or, if there is no such assignment, -1). The difference is that check searches from the front to the middle of the building, while check_reflect searches from the back towards the middle of the building. The left shape A is in effect reflected.

```javascript
function check (shape) {
    for (i = shape.purlin_front; i <= (shape.purlin_mid-1); i++) {
        // from the front to the middle
        if (shape.purlin_var[i] == "circ_sol_show" and
            shape.purlin_var[i + 1] <> "hide" and
            // the "next" position is towards the back
            shape.column_var[i] == "show" and
            shape.beam_var[i].indexOf ("show") <> -1 and
            shape.stage_var == "B_show") {
            return i;
        }
    }
    return -1;
}
```

```javascript
function check_reflect (shape) {
    for (i = shape.purlin_back; i >= (shape.purlin_mid + 1); i--) {
        // from the back to the middle
        if (shape.purlin_var[i] == "circ_sol_show" and
            shape.purlin_var[i + 1] <> "hide" and
            // the "next" position is towards the front
            shape.column_var[i] == "show" and
            shape.beam_var[i].indexOf ("show") <> -1 and
```
4.4. PREVIEWING THE NEXT DESIGN

When the user rolls over an applicable (undimmed) schema, the next design is created and displayed. The rollover code for schema A2 is shown below.

```
on (release, rollOver) {
    // init begin
    _root.current_rule = "A2";
    if (/:display_mode == "normal") {
        _root.current_stage = "normal/initshape";
        _root.stage_preview = "normal/preview";
        _root.rule_object = "normal/arrow";
    } else if (/:display_mode == "overview") {
        _root.current_stage = "overview/derivation/stage" + String(_root.history.stage_current);
        _root.stage_preview = "overview/derivation/stage" + String(_root.history.stage_current + 1);
        _root.rule_object = "overview/derivation/arrow" + String(_root.history.stage_current + 1);
    }
    _root.show_rule (_root.current_rule, _root.rule_object);
    // init end
    target = check (_root.shape_current);
    show_match (_root.shape_current, target);
    _root.shape_new = apply (_root.shape_current);
    _root.preview_change (_root.shape_new, _root.stage_preview);
    show_change (_root.shape_new, target);
    /:change = 1;
    gotoAndStop ("highlight");
}
```

The important functions here are check, show_match, apply, preview_change, and show_change. Check has already been seen above.

**Show_match highlights g(f(A)) as a part of C.** It accepts as its arguments the current shape shape_source and the assignment target. It makes a copy shape of the current shape and, for each attribute of each element in g(f(A)), overwrites the value as highlight.

```
function show_match (shape_source, target) {
    shape = new Object();
    _root.Object_duplicate (shape_source, shape);
    // shape.purlin_var[target] = "circ_hol_highlight";
    shape.purlin_var[target + 1] = "circ_hol_highlight";
    shape.column_var[target] = "highlight";
    shape.control_var[target] = "tri_sol_highlight";
    shape.control_var[target + 1] = "circ_hol_highlight";
    shape.baseline_var[target] = "highlight";
    shape.baseline_var[target + 1] = "highlight";
    shape.stage_var = "A_highlight";
}
```
In addition, show_match calls show_matching, which activates the display. Show_matching takes as its arguments the next shape shape and the variable current_stage, which specifies the view (current transformation or derivation). It examines the value of each attribute of each element; if the value is highlight, then the element is highlighted.

After calling show_match, the rollover code calls apply, which takes as its argument the current shape shape_source and returns the next shape shape. It does this by creating a copy of the current shape and transforming it through addition and subtraction. The code for apply is shown below.

```javascript
function apply (shape_source) {
  shape = new _root.Shape();
  _root.Object_duplicate (shape_source, shape);
  //
  shape = Add (Subtract (shape, rule_left), rule_right);
  shape.c++;
  if (shape.be2 <> "∅") {
    shape.be2 += chr(13) + "1-rafter beam in front";
    shape.bc2 += chr(13) + "qian_zhaqian";
  } else {
    shape.be2 = "1-rafter beam in front";
    shape.bc2 = "qian_zhaqian";
  }
  shape.be3 = "with " + String (shape.c) + " columns";
  shape.bc3 = "yong_" + String (shape.c) + "_zhu";
  return shape;
}
```

The important functions in apply are Add and Subtract, which we will return to shortly. The next descriptions are created by simply inserting new text strings into the current descriptions.

Preview_change takes as its arguments the next shape shape and the variable shape_name, which specifies the view (current transformation or derivation), and displays the next shape in the appropriate view.

Show_change is similar to show_match: it highlights \( g(t(B)) \) as a part of \( C(i+1) \).

### 4.5. ADDITION AND SUBTRACTION

As mentioned above, rule_left is created by init_rule and intuitively is the “inverse” of \( g(A) \), that is, \( -g(A) \). Subtract and Add are identical. Add takes as its arguments two shapes shape_A and shape_B and returns their sum shape. It examines each attribute of each element of shape, and
overwrites that value onto the corresponding value in \texttt{shape\_B}, unless that value is \texttt{none}. The code for \texttt{Add} is shown here.

\begin{verbatim}
function Add (shape\_A, shape\_B) {
  shape = new _root.Shape();
  _root.Object_duplicate (shape\_A, shape);
  //
  for (i = shape.purlin\_front; i <= shape.purlin\_back; i++) {
    if (shape\_B.purlin\_var[i] <> "none") {
      shape.purlin\_var[i] = shape\_B.purlin\_var[i];
    }
    if (shape\_B.column\_var[i] <> "none") {
      shape.column\_var[i] = shape\_B.column\_var[i];
    }
    if ((shape\_B.beam\_var[i] <> "none") and (i < shape\_A.purlin\_back)) {
      shape.beam\_var[i] = shape\_B.beam\_var[i];
    }
    if (shape\_B.roof\_var[i] <> "none") {
      shape.roof\_var[i] = shape\_B.roof\_var[i];
    }
    if (shape\_B.control\_var[i] <> "none") {
      shape.control\_var[i] = shape\_B.control\_var[i];
    }
    if (shape\_B.controlclean\_var[i] <> "none") {
      shape.controlclean\_var[i] = shape\_B.controlclean\_var[i];
    }
    if (shape\_B.baseline\_var[i] <> "none") {
      shape.baseline\_var[i] = shape\_B.baseline\_var[i];
    }
  }
  if (shape\_B.vertaxis\_var <> "none") {
    shape.vertaxis\_var = shape\_B.vertaxis\_var;
  }
  if (shape\_B.controlfront\_var <> "none") {
    shape.controlfront\_var = shape\_B.controlfront\_var;
  }
  if (shape\_B.controlback\_var <> "none") {
    shape.controlback\_var = shape\_B.controlback\_var;
  }
  if (shape\_B.stage\_var <> "none") {
    shape.stage\_var = shape\_B.stage\_var;
  }
  return shape;
}
\end{verbatim}

5. Conclusion

We have seen the implementation of a parametric set grammar that supports neither emergence nor matching under multiple transformations. The infelicities of this implementation are evident and many: it is complicated, inefficient, and inconsistent with formal definitions.

On the other hand, we know from classroom experience that it does what we designed it to do, namely to show the current design, check the applicability of a schema, preview the next design, and apply a schema. These four capabilities have produced an interface that has proven to be
virtually self-explanatory. In fact, rather than using shape grammar to explain the interpreter, we now use the interpreter to explain shape grammar.

That a set-based implementation, even a suboptimal one such as this, can accomplish so much suggests that we have not addressed the really difficult issues, chief among which is, according to Chase (2002, 162), “handling the unexpected nature of emergent features.” Indeed, it seems clear that the key to easy implementation is simply to avoid emergence altogether.

The question is begged: can emergence be avoided? We believe that it depends on the expected interaction between user and implementation. If the grammar is not expected to be static – if, for example, the user is allowed to revise it – then emergence is indispensable. An infallible set representation is impossible, because it is impossible to foresee all possible revisions.

This is exactly what happened in our classroom experience. Recall that we had expected that students would modify the grammar by increasing the constraints on schema application; we did not expect students to alter other aspects of the grammar. In the event, some students modified the schemata (on paper) in ways that we had not anticipated – for instance, inserting columns in unexpected positions – and that therefore could not be supported by our set representation.

Our implementation fell short, given our premise that the grammar is a hypothesis to be tested and revised. In addition, we had made no provision for students to modify the interpreter by altering existing rules or defining new ones. This is an interface challenge that we thought we had been spared.

But if, on the other hand, the user simply “operates” (and does not modify) the implementation – that is, he uses the grammar as is – then an appropriate set representation will suffice. Thus Flemming (1987) does not miss a general interpreter for two reasons: his set representation is immune from challenge, and his interface does not need to support rule definition by users.

It seems that we can forego emergence with only a narrow category of grammars: those that are fixed or changeable within known limits. For all others, including ours, which we had initially considered analytic, we must be able to implement emergence. This is one research goal.

At the same time, we believe that, between the limitations of a fixed representation and the complexity of emergence, there may be room for creative expedience. It would be worthwhile to investigate further the relation between the user–interpreter interface and the technical characteristics of the implementation.

Acknowledgements

We would like to thank the Chinese University of Hong Kong (CUHK) for a Direct Grant for Research; the Department of Architecture, CUHK, for
special support; Wang Yang for preparing the illustrations; and our students for cheerfully taking the grammar in directions we had not foreseen.

References


Li, AI: 2001, A shape grammar for teaching the architectural style of the Yingzao fashi, Ph.D. dissertation, Department of Architecture, Massachusetts Institute of Technology, Cambridge, Mass.


Li, AI: 2003, The Yingzao fashi in the information age, paper read at The Beaux-Arts, Paul-Philippe Cret, and 20th-century architecture in China, at University of Pennsylvania.

Li, AI: forthcoming, Styles, grammars, authors, and users, paper read at Design computing and cognition ’04, at Cambridge, Mass.

Liang Sicheng: 1984, Zhongguo jianzhu zhi liangbu “wenfa keben” [The two “grammar books” of Chinese architecture], *Liang Sicheng wenji* [The collected works of Liang Sicheng], Zhongguo jianzhu gongye, Beijing, pp. 357–363.


Stiny, G: 1990, What is a design?, *Environment and planning B: planning & design* 17: 97–103.

Abstract. We consider an extension of the algebraic framework for shape grammars to various information types. We denote this framework sorts. We describe an implementation of sorts in Java.

1. Introduction

Grammar formalisms have been around for over 40 years and have found application in a wide variety of disciplines and domains, to name a few, natural language, architectural design, mechanical design, and syntactic pattern recognition. Their implementations, however, have been mostly narrowly focused and sparse. In design, in particular, the expectation of grammar formalisms or similar rule-based systems to pervade design software has so far remained only an illusion. There are three main reasons for this. The first relates to the difficulty stemming from technical considerations of implementing grammars. The second difficulty pertains to ways of enabling designers to employ grammatical rules in a manner that does not impede their act of designing. The third difficulty affects the rapid development, adaptation, and maintenance of grammar-based systems.

Grammar formalisms come in a large variety, requiring different representations of the objects being generated, and different interpretative mechanisms for this generation. At the same time, all grammars share certain definitions and characteristics. Grammars are defined over an algebra of objects, $U$, that is closed under the operations of addition, $+$, and subtraction, $-$, and a set of transformations, $F$. In other words, if $u$ and $v$ are members of...
\( U \), so too are \( u + f(v) \) and \( u - f(v) \) where \( f \) is a member of \( F \). In addition, a match relation, \( \leq \), on the algebra governs when an object occurs in another object under some transformation, that is, \( f(u) \leq v \) whenever \( u \) occurs in \( v \) for some member \( f \) of \( F \), if \( u \) and \( v \) are members of \( U \).

Computer implementation of shape grammars has been of interest for some considerable time. Most implementations of grammar interpreters apply to two dimensional shapes (Gips 1975; Krishnamurti 1982; Krishnamurti and Giraud 1986; Chase 1989; Tapia 1996; 1999; McCormack and Cagan 2002). Few have been implemented for three dimensional shapes and, mostly, these are restricted to certain kinds of shapes (Piazzalunga and Fitzhorn 1998), or based on representations that do not readily provide for ‘emergent’ or unanticipated subshape relationships (Longenecker and Fitzhorn 1991; Heisserman 1991; 1994), or were developed for specific purposes (see for example, Flemming 1987; Agarwal and Cagan 1998).

The technical machinery required for shape grammar implementations has, to a large extent, been established. Krishnamurti and Stouffs (2004) present a unified approach for arithmetic in any shape algebra, including curves and surfaces. Krishnamurti and Earl (1992) investigate shape recognition in \( U_{13} \); Krishnamurti and Stouffs (1993; 1997) consider, respectively, shape recognition in \( U_{23} \) and in the cartesian product \( U_0 \times U_1 \times U_2 \times U_3 \). The notation \( U_i \) refers to linear shapes made up of \( i \)-dimensional elements in \( j \)-dimensional Euclidean space, and \( U_i \) is shorthand when the dimensionality of the space is known (Stiny 1991).

However, most practical problems of generative design are not limited to geometry only. Part of the attractiveness of the algebraic model underlying shape grammars is its ability to include non-geometric attributes, such as labels (Stiny 1980; 1990), weights (Stiny 1992) and colors (Knight 1989). These augmented shapes are derived from shapes of spatial elements by associating symbols, labels or properties to the elements. Consequently, their algebraic operations are redefined in order to deal correctly with the associated symbols. The result is a different grammar formalism each time the attribute type is altered, which does not support the rapid development, adaptation, and maintenance of grammar-based systems.

Instead, by considering algebras not only for shapes but for many different types of information and by considering compositional operations on algebras, a framework can be established that enables the exploration of different grammar formalisms, based on a variety of algebras and interpretative mechanisms. \textit{Sorts} (Stouffs and Krishnamurti 2002) implements such a framework. \textit{Sorts} constitute a model for representations that defines formal operations on \textit{sorts} and recognizes formal relationships between \textit{sorts}. Each \textit{sort} defines an algebra over its elements; formal compositions of \textit{sorts} derive their algebraic properties from their component
Grasses, sorts and implementation

Sorts. As such, sorts enable the development of alternative representations of a same entity or design, the comparison of representations with respect to scope and coverage, and the mapping of data between representations, as well as data recognition and the specification of design rules.

Sorts can be considered as class structures identified by compositions of named data entities (Stouffs et al. 1996). These data entities are identified by a type specifying the set of possible values. Exemplar types are labels and numeric values or weights, and spatial types such as points, line segments, plane segments and volumes. Data entities are composed or grouped using one or more constructors, these are devices for relating entities together. At this time, we consider two constructors, resulting in either a subordinate composition of properties or a disjunctively co-ordinate composition. Others can be defined.

We are implementing sorts using an object-oriented approach (in Java). This modular approach enables the inclusion of new entity types (and constructors) without any modifications to the rest of the code. We distinguish three major types of object classes (figures 1 and 2). Individuals define the data entities (The term individual refers to Stiny’s treatise of shapes as individuals (1982)). Forms are collections of individuals from the same sort; the object class defines the respective constructor. Sorts define the class structures.

2. Individuals and data operations

Grammars rely on a match relation to be defined on each algebra, which governs when an object occurs in another object under some transformation. This partial order relation is crucial to all data, both individuals and forms of individuals. The ability to compare individuals as to whether one is less than, greater than, less than or equal to, or greater than or equal to another is encoded in the Element class which is an abstract super class to both the classes Individual and Form (figure 1).

Each algebra must further specify the operations of sum and difference (and product). These operations on forms (or collections) of individuals can be expressed in terms of operations of combine and complement of one individual with respect to another. Corresponding methods are defined in the abstract Individual class and inherited or overwritten by each subclass, where each subclass specifies a particular data entity type. Currently, the following entity types are implemented: alphanumeric (labels) and numeric values and (numeric) weights, points, (infinite) lines, line segments and (infinite) planes, circles and (circular) arcs, URLs for texts and images, and unique IDs, (positive or negative) signs and property relationships. Below, we present a few exemplar entity types in detail.
Figure 1. UML diagram depicting the abstract classes Element, Individual and Form and a number of exemplar subclasses, such as Label, Point, OrdinalForm and DiscreteForm.

Figure 2. UML diagram depicting the abstract class Sort and three exemplar subclasses, PrimitiveSort, AttributeSort and DisjunctiveSort.
A sort of labels defines a label as a string of characters. In the algebra of labels, the empty label (string) defines the neutral element. Then, two labels combine if and only if both labels are identical, and the complement of a label with respect to another label is the empty label, if both labels are identical, and is the first label, otherwise.

A sort of (numeric) weights defines a weight as a positive (floating-point) value. 0 defines the neutral element in this algebra. Two weights always combine to form a single weight, the value of which is the maximal value of both weights. The complement of one weight with respect to another is 0, if this weight’s value is less than or equal to the other, and is the weight itself, otherwise.

In the case of geometric data entities, an individual is always represented by a carrier, or co-descriptor, and, possibly, a boundary. Two individuals of the same geometric algebra combine only if they have the same co-descriptor and are not disjoint. Two individuals with equal co-descriptor are said to be disjoint if they do not overlap (do not share a common part) and they do not touch (do not share a common boundary). The abstract class Geometry defines methods to compare co-descriptors and to determine whether two individuals are disjoint, touch or are aligned (if they share a common part and a common boundary) (figure 1). This class is the super class for all geometric individual classes.

A sort of points defines a point as a vector of (rational) coordinates. A nil individual is defined to be the neutral element for this algebra. The co-descriptor of a point is the point itself, a point has no boundary. Similarly to labels, two points combine if and only if both points are identical, and the complement of a point with respect to another point is nil, if both points are identical, and is the first point, otherwise.

A sort of (infinite) lines defines a line as a couple of vectors. The first vector defines the direction of the line, the second the root point of the line (the intersection point of the line and a perpendicular line through the origin). Again, a nil individual is defined to be the neutral element for this algebra. The co-descriptor of a line is the line itself, an infinite line has no boundary. Similarly again, two lines combine if and only if both lines are identical, and the complement of a line with respect to another line is nil, if both lines are identical, and is the first line, otherwise. The class Line serves as a super class for the class LineSegment. A sort of (bounded) line segments defines a line segment as a line with, additionally, a couple of position vectors. This line defines the co-descriptor of the line segment, the position vectors define the boundary positions of the segment. Two line segments combine if and only if their co-descriptors are identical and they are not disjoint. The complement of a line segment with respect to another line segment is a new line segment, if both co-descriptors are identical and the
line segments are not disjoint and do not touch, and is the first line segment, otherwise. The new line segment is defined in terms of the boundary positions of both line segments (see below).

3. Forms and data behaviors

The abstract class Form defines the operations of sum and difference (and product), and if one form is part of another form. These operations can be defined in terms of (among others) the operations combine and complement. However, this relationship is dependent on the way the operations of combine and complement are defined and thus on the type of individuals. For example, the operations of combine and complement are similarly defined for sorts of labels, points and (infinite) lines (see above). We denote this relationship between the algebraic operations of sum and difference and the operations of combine and complement on individuals the (operational) behavior of the sort.

The behavior of the sorts of points and labels is a discrete behavior, corresponding to mathematical set operations. In this case, the part relation is defined as a subset relation, and the operations of sum, difference and product correspond to set union, difference and intersection, respectively. In other words, if \( a \) and \( b \) denote two forms of a sort with discrete behavior, and \( A \) and \( B \) denote the corresponding sets of data entities (e.g., points or labels), then (\( a : A \) specifies \( A \) as a representation of \( a \))

\[
\begin{align*}
\ a : A \land b : B & \Rightarrow a \leq b \Leftrightarrow A \subseteq B \\
\ a + b : A \cup B \\
\ a - b : A / B \\
\ a \cdot b : A \cap B
\end{align*}
\] (1)

Weights adhere to a different behavior. Considered weights to denote thickness for points and lines (or tones for surfaces and volumes). Then, a behavior for weights becomes apparent from drawings: a single line drawn multiple times, every time with different thickness, appears as it was drawn once with the largest thickness, even though it assumes the same line with other thickness (Stiny 1992). This behavior is termed ordinal; the part relation on weights corresponds to the less-than-or-equal relation on numbers;

\[
\begin{align*}
\ a : \{x\} \land b : \{y\} & \Rightarrow a \leq b \Leftrightarrow x \leq y \\
\ a + b : \{\max(x, y)\} \\
\ a - b : \{\} \text{ if } x \leq y \text{, else } \{x\} \\
\ a \cdot b : \{\min(x, y)\}
\end{align*}
\] (2)

An interval behavior applies to line segments (as well as intervals of time or other one-dimensional quantities). A specification of the interval behavior
can be expressed in terms of the behavior of the boundary (positions) of the interval. Let $B[a]$ denote the boundary of a form $a$. In the case of a form of line segments, the boundary of this form is the collection of boundary positions from all line segments. This boundary can be partitioned with respect to another form $b$ of line segments by distinguishing those boundary positions that lie within a line segment from $b$, those that lie outside of all line segments from $b$, and those that belong to the boundary of $b$. Let $I_a$ denote the boundary of $a$ that lies within $b$ and $O_a$ denote the boundary of $a$ that lies outside of $b$. Let $M$ denote the shared boundary of $a$ and $b$ where the respective line segments are aligned, and $N$ the shared boundary of $a$ and $b$ where the respective line segments touch (Stouffs and Krishnamurti 2004; figure 3). Then,

$$a : B[a] \land b : B[b] \Rightarrow a \leq b \iff I_a = 0 \land O_b = 0 \land N = 0$$

$$a + b : B[a + b] = O_a + O_b + M$$

$$a - b : B[a - b] = O_a + I_b + N$$

$$a \cdot b : B[a \cdot b] = I_a + I_b + M$$

(3)

Figure 3. The specification of the boundary collections $I_a$, $O_a$, $I_b$, $O_b$, $M$ and $N$, given two collections of intervals $a$ (above) and $b$ (below).

Similar behaviors can be specified for plane segments and volumes (Stouffs and Krishnamurti 2004) as well as hypersegments of higher dimension; (3) still applies though the construction of $I_a$, $O_a$, $I_b$, $O_b$, $M$, and $N$ is correspondingly more complex (figure 4).

4. Sorts and data compositions

So far, we have considered only sorts that are made up of a single type of data entities, such as sorts of labels, points or line segments. We denote these primitive sorts. Primitive sorts can be combined into composite sorts under formal compositional operations. Examples of composite sorts are labeled points and weighted line segments, but also the combination thereof. At this time, we consider two formal operations for composing sorts. The attribute operator specifies a subordinate composition of sorts. Under the attribute operator, an individual of the resulting sort is an individual of the first operand sort that has as attribute a form of the second operand sort. For
example, a form of the sort of labeled points specifies a collection of points, where each (individual) point has a form of labels assigned as attribute.

\[
labeled\_points : \text{points} \uparrow \text{labels}
\] (4)

![Figure 4. The boundary collections \( I_a, O_a, I_b, O_b, M \) and \( N \) for two volumes \( a \) and \( b \), and the collections of volumes resulting from the operations \( a + b, a \cdot b, a - b \) and \( b - a \).]

The operation of sum allows for disjunctively co-ordinate compositions of multiple sorts. Under the operation of sum, a form of the resulting sort is a combination of forms of each of the operand sorts. For example, a form of the sort of labeled points and line segments specifies a collection of labeled points and/or line segments. Such a form may contain only labeled points, only line segments, or both labeled points and line segments.

\[
\text{shapes} : \text{linesegments} \uparrow \text{labeled\_points}
\] (5)

4.1. COMPOSITE BEHAVIORS

A composite sort defines its own algebra, which is a composition of the operand algebras in relationship to the formal compositional operator that defines the composition. As such, a composite sort can also be considered to define a behavior. This behavior is inherited from the component sorts in a manner that depends on the compositional relationship. Under the operation of sum, the behavior is that of the component sort for each component. Forms from different component sorts never interact, the resulting form,
corresponding the composite sort, is the collection of forms from all component sorts. When an operation applies to two forms of the same composite sort, the operation instead applies to the respective component forms.

The attribute operation on sorts specifies a dependency relation on the sorts in a composition, where each component, except the first, defines an attribute sort to the previous component. That is, a corresponding form consists of individuals of the first component sort, each element of which has, as attribute, another form corresponding to the sort as a composition of all but the first component, in a recursive manner. Thus, the behavior of such a sort is defined by the behavior of its first component sort. Specifically, when an operation applies to two forms of the same composite sort (under the attribute relationship), identical individuals combine and their attribute forms are composed under the same operation. Any resulting individuals that have an empty attribute form are removed.

4.2. SORT CLASSES

The abstract class Sort defines a generalized sort (figure 2). For efficiency purposes, a partial order is defined on sorts that allows these to be compared as to whether one is less than, greater than, less than or equal to, or greater than or equal to the other. A sort can be provided a name, sorts can be combined under the attribute operation or the operation of sum, and sorts can be compared as to whether one contains another, or vice versa, one is part of another.

Primitive sorts are distinguished by their name and their type (figure 2). This type is denoted the characteristic individual of the sort and is represented by the respective class. Primitive sorts have their behavior specified as part of their characteristic individual. A static method in the class PrimitiveSort allows each class of individuals to register itself and specify a behavior (or class of forms) at the same time. In this way, classes of individuals can easily be added without any need to alter or recompile the rest of the class package. Programming wise, they are only ever identified by their class (characteristic individual) and their behavioral class.

Attribute sorts are distinguished by their base and weight sorts. The base sort specifies the sort of individuals, the weight sort specifies the sort of attribute forms. Disjunctive sorts are distinguished by a list of (disjunctive) component sorts. Both attribute sorts and disjunctive sorts may be specified a name. The representational composition of sorts is subject to rules of reduction, in order to ensure a semi-canonical form (Stouffs and Krishnamurti 2002). For example, the operation of sum on sorts distributes over the attribute operation, such that a base sort is never a disjunctive sort:

\[(a + b) \land c = (a \land c) + (b \land c) \] (6)
We are currently developing a prototype interface to build and edit definitions of sorts, compare and match sorts and to construct corresponding forms, in order to investigate the interaction with sorts, especially when these become large structures.

Acknowledgements

The first author is funded by a grant from the Netherlands Organization for Scientific Research (NWO), 016.007.007, support for which is gratefully acknowledged. The second author is funded by a grant from the National Science Foundation, CMS #0121549, support for which is gratefully acknowledged. Any opinions, findings, conclusions or recommendations presented in this paper are those of the authors and do not necessarily reflect the views of the Netherlands Organization for Scientific Research or the National Science Foundation. The authors would like to thank Michael Cumming for his work on the development of a prototype interface to build and manipulate sorts.

References


SHAPER 2D

An Interactive Shape Grammar Interpreter

MIRANDA C MCGILL
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract. Shaper2D is a program that promotes the use of computers for learning about computational design by employing a dynamic, interactive interface.

1. Introduction

Shaper2D is a program that promotes the use of computers for learning about computational design. This program, created by a designer rather than a programmer, was developed to employ an intuitive, visual interface that encourages a “learning by designing” approach to shape grammar education.

Shaper2D is not the first application created for shape grammar exploration. Examining two previous computer implementations of shape grammars situates Shaper2D in its intended context. These software applications for exploring two-dimensional and three-dimensional shape grammars provide powerful tools for learning fundamental concepts, but their interfaces are not conducive to experimenting with shape grammars. While these programs demonstrate simple and accurate representations of the actual shape grammar computation, they do not provide an appropriate interface for designers.

2. Designing Design-Mediating Software

The development of Shaper2D responded to the need for an improved human-computer interface for exploring shape grammars. It has been widely agreed that hand computation is essential for a comprehensive understanding of the concepts involved in shape grammars. However, using the computer can facilitate designing with shape grammars, as designs tend to get very complicated very quickly. Shaper2D makes the shape grammar process as transparent as possible through interactivity and dynamic response.
Designing becomes more comprehensible; designers are encouraged to persevere with an idea rather than give up due to frustration or impatience. *Shaper2D* capitalizes on the advantages of computer power over hand computation, especially when it comes to more complex design generation.

2.1. A PROGRAMMER’S APPROACH TO PROGRAMMING

It is important to consider how a programmer develops a program, in order to contextualize the different approach taken with *Shaper2D*. While the focus on an intuitive interface and relevant functionality may seem an obvious approach to designing an application, this is not always the case. Often the “worse is better” philosophy, as discussed by Richard Gabriel, is used in software development.

“…It is more important for the [program’s] implementation to be simple than the interface. Simplicity [of code] is the most important consideration in a design.” (Gabriel, 1999)

While not all programmers fall prey to this idea, it is still the most typical approach. Essentially, “worse is better” programming sacrifices the interface and usability for simplicity of code design, that is:

- Simple code is more important than a usable interface, in other words, the programmer’s convenience is more important than the user’s convenience.
- The user is expected to learn to think about the problem the way the programmer thinks about the problem.
- The interface should reflect the internal structures of the program since they are the most fundamental description of the problem.

This approach usually results in a program with enormous functionality, which is difficult to use and has an unintuitive interface.

When a user has difficulty using a program, it is typical that they blame themselves rather than look to the inadequacies of the software they are using. This should not be the case.

2.2. A DESIGNER’S APPROACH TO PROGRAMMING

2.2.1. Interface

From the outset, *Shaper2D* was designed with the end user in mind. Rather than programming an application and then tacking on an interface at the end, developing a dynamic, intuitive, and simple interface was the driving force behind the program design. Three rules were steadfastly adhered to:

- Never choose to sacrifice the usability of the program for the sake of making the code simpler.
• If a help manual is required for learning to use the program then the interface is flawed.
• Remember that the end user is a designer not a programmer so the program should be intuitive, visual, and responsive.

Interfaces developed by programmers have a tendency to mirror the internal structure of a program without consideration for the end user. This approach is borne out of convenience and often results in powerful programs that sometimes include a token interface as an acknowledgment for the end user, but which usually require a lengthy help manual in order to understand how to run them.

2.2.2. Functionality

The deliberate restrictions placed on Shaper2D could be described as additional functions in their own right. Shaper2D was designed to force the user to use the program in a certain way in order to learn basic shape grammars. Ensuring simplicity of use made the interface design more problematic and restricting the program’s functionality, such as forcing the user to manipulate the rule geometry and labels separately, complicated programming.

Programmers are often reluctant to restrict what a program can do, regardless of whether a simpler, less feature-heavy—or restricted—program would be more beneficial to the end user.

3. A New Shape Grammar Interpreter

3.1. PREVIOUS SHAPE GRAMMAR INTERPRETERS

To frame the motivation behind the development of Shaper2D, it is helpful to consider two recent shape grammar interpreters, 3D Shaper (Wang 1999) and GEdit (Tapia 1999). These programs were developed to expedite the process of design using shape grammars and have been used on occasion for classroom teaching. However, their use in the design studio has been limited. Various reasons have been cited for this, including a completely unrestricted workspace that is only truly useful to more advanced uses of shape grammars, delayed feedback, and a designer-unfriendly interface. These are not criticisms of the software, as each application has an audience in current shape grammar pedagogy and research. However, the deficiencies listed above are those that Shaper2D seeks to address.

3.1.1. 3D Shaper

3D Shaper was developed by Yufei Wang in 1999 to simulate the designs produced by manipulating wooden “Froebel” blocks. Compiled for the
UNIX/SGI operating system, it is a very powerful program that enables the user to experiment with three-dimensional grammars in ways that are difficult to visualize mentally, and often impossible to perform physically.

3D Shaper implements a static interface that requires the user to type in numerical parameters to determine the size and type of shapes, as well as the spatial relation between the shapes. The program then applies the rules and produces an Open Inventor file containing the three-dimensional design output. The user opens this file in an appropriate viewer for review. However, if a design change is needed then the user must return to the 3D Shaper window, input the necessary shape and/or rule revisions, and run the program again in order to generate a new Open Inventor file.

3.1.2. GEdit
The first computer implementation of shape grammars that included a viable user interface was the Apple Macintosh-based GEdit developed by Mark Tapia (1999). This interpreter is used for generating two-dimensional, non-parametric shape grammars and emphasizes that the “...drawing is the computation” (Tapia 1999). Tapia sought to minimize user distraction, both in terms of obscuring parts of the program and perverting the design flow, by limiting the use of drop-down menus and dialog boxes. He instead implemented an object-specific radial menu, which only appears when needed to ensure that the user’s focus is not removed from the design process. This is a program for shape grammar experts, since it is very open-ended and allows for the free exploration of almost any kind of non-parametric, two-dimensional shape grammar.

3.2. WHAT IS DIFFERENT ABOUT SHAPER2D?
Previous interpreters developed to expedite the process of design using shape grammars lacked an accessible user interface, were operating system-specific, were too general, or did not account for the dynamic, interactive environment sought by a designer.

With 3D Shaper, the process of first inputting numbers into a static interface then opening up a separate viewer in order to see the design does not tally with the designer’s habitual design process. While the parameters entered by the user fully describe the desired grammar, they do so in a way reflective of the internal computations of the software as opposed to the way the designer would manipulate shapes naturally. The interface was written for the ease of the programming, not for the benefit of the user. For many architects, conceptual designing is fluid and dynamic—3D Shaper is an invaluable tool for experimenting with three-dimensional shape grammars, but the interface is not analogous to the design process and hence does not facilitate quick, experimental design.
GEdit, while introducing the notion of an elegant interface, is a generalized, unrestricted example of shape grammar software. It is suitable for advanced shape grammarians wishing to visualize a particular derivation, but is impractical for novice users who require a more constrained, structured environment or for users wishing to use shape grammars for real-world design problems.

Shaper2D is a visually dynamic shape grammar application for generating designs using very restricted kinds of shape grammars. Changes to the grammar are immediately reflected in the design. It was written to overcome the platform-specific limitations imposed by previous interpreters. The application has been run successfully under many major operating systems (Microsoft Windows, Mac OS and Linux), and the applet runs under any web browser capable of running Java™2 (such as Mozilla, Internet Explorer, and Opera).

4. The Development of Shaper2D

4.1. INTERFACE AND INTERACTIVITY

The interface is the key aspect of Shaper2D’s development. It had been noted by several researchers in the field of shape grammars that while useful implementations of shape grammars have been developed, little attention had been focused on the interface (March, 1997; Knight, 1998; Tapia, 1999). This was a surprising oversight given the anticipated audience.

Visual seductiveness is very important to designers. So naturally, when developing Shaper2D, many graphic and interface design issues had to be considered. In particular, the interface needed to embrace the inherent visual uniqueness of shape grammars. The decision to exclude any need for manual typing-in of parameters or instructions was made at the program’s inception—it was important to develop a program based on an interactive, visual interface to generate designs.

The most comprehensible way of doing shape grammars is when the computation is done by hand—indeed, hand computation using tracing paper played an important role in the development of Shaper2D. However, because the actual computations are concealed the program could be described as a “black box”: the rule application process is hidden from the user. Physically flipping and rotating drawings on tracing paper, or manipulating three-dimensional wooden blocks, personalizes the act of performing the spatial transformations by exposing the designer to direct contact with the processes involved.

The dynamic interface endeavors to offset the inherent black box disadvantages by informing the designer of the effect of her actions immediately. The user can experiment quickly with different rules and
iterations in order to compare the outcomes of different ideas and grammar applications.

4.2. HOW SHAPER2D WORKS

*Shaper2D* was developed as both a Java applet (figure 1) and a stand-alone application (figures 2 and 3). This decision was based on the advantages of using the Internet for enhancing portability and cross-platform compatibility.

The *Shaper2D* user interface is composed of a menu bar, five panels—“Spatial Relation” (1 and 2), “Rule” (1 and 2), and “Design”—and two toolbars for selecting different shapes. The default shapes are a rectangle, square, equilateral triangle, and isosceles triangle. The “Spatial Relation 2” and “Rule 2” panels are grayed out by default when the program is first run to emphasize the difference between using one rule and two rules.

Whenever the user manipulates a shape—which can be done in both the “Spatial Relation” and “Rule” panels—the other active panels update in real-time to reflect the change. This enables the user to get a dynamic response to her actions. Similarly, when the position of a label is changed the “Design” panel updates immediately.

![Figure 1. The Shaper2D Applet (version 1.0)]
The dynamic update is a key feature of the program and enables the user to see the implications of each design move instantaneously. The “Design” panel updates dynamically according to changes made in the “Spatial Relation” and “Rules” panels. Additionally, to review the construction of a design, the user has the option to display the labels (figure 4)—with the most recent shape addition highlighted in red. The label display feature is particularly helpful when the user gradually increments the number of iterations displayed, in other words, performs a 'walk through' of a design's construction.
4.2.1. Shaper2D Applet

The advantage of the applet (http://www.mit.edu/~miri/shaper2d) is that it can be run directly from an Internet browser. This allows users who may not be able to download and run the application to have access to the software. However, this version has limited functionality due to the security restrictions imposed on applets.

4.2.2. Shaper2D Application (Basic & Advanced)

The Shaper2D application is a more feature-rich tool than the applet. The main difference between the applet and application is the ability to save and load: the user is able to save and load rules and can export a design in the DXF file format for importing into another CAD application (such as AutoCAD), thus increasing the pedagogical and design value of the software.

The user is also able to import a background image into the design window, in order to place a design in a given context. In turn, the user can change the line color of the design to improve visibility (figure 5).

Two versions of the application were created, one for novices and the other for more experienced users. The decision to develop two versions was in response to Shaper2D being designed as a pedagogical tool. Rather than overwhelm beginners with an over-complex interface, an advanced version was created which includes additional features. Thus, when the user is confident enough to progress to more complex shape grammar investigations she can change from the basic to the advanced interpreter.
4.2.3. Shaper2D Application (Advanced)

This version has the advantage of allowing shape substitution (figure 6). This feature enables the user to import a substitute shape or design to be used in place of one of the four preexisting, or “reference”, shapes. The substituted shape retains the symmetry group of the reference shape, thus allowing the user to explore the implications of different symmetries. More complex designs can be developed providing the user an opportunity to investigate how the different symmetries of the reference shape affect the rule application.
5. Conclusion

*Shaper2D* was developed by a designer rather than a programmer, and strives to promote a more fluid approach to software design for designers. Previous shape grammar programs—designed by programmers—do not offer the user the opportunity to quickly explore many designs, so fail to offer any advantages over hand computation. They tie the user into the “plan, calculate, examine, redesign” method of designing—in other words, an iterative and algorithmic approach—where each stage is seen as a distinct step. The user loses the ability to explore and experiment.

*Shaper2D* encourages exploration by providing the user with instant feedback and a purely visual, designer-friendly interface. Designing becomes more comprehensible; designers are encouraged to persevere with an idea rather than give up due to frustration or impatience. *Shaper2D* capitalizes on the advantages of computer power over hand computation, especially when it comes to more complex design generation.

Acknowledgements

Thanks to: Terry Knight, Oliver Dial, William Mitchell, Edith Ackermann, Mark Tapia, and everyone who has taken the time to test *Shaper2D* and provide me with valuable feedback.

References


FROM SIMPLE TO COMPLEX

Using AutoCAD to build generative design systems

GABRIELA CELANI
UNICAMP – Universidade Estadual de Campinas
Cidade Universitária “Zeferino Vaz” – Distrito de Barão Geraldo
Faculdade de Engenharia Civil
Departamento de Arquitetura e Construção
Campinas – SP – Brazil – CEP 13083-970
celani@fec.unicamp.br | celani@alum.mit.edu

Abstract. The present paper aims at describing the use of an “over the counter” CAD tool – AutoCAD – to build from very simple to reasonably complex generative design systems.

1. Introduction

In principle, computational design methods not necessarily need to be implemented in the computer. However, such implementation has the advantage of automating the repetitive, tedious work, and thus allowing for the exploration of a larger number of design alternatives in a short period of time, both in the case of combinatorial generative design and in the heuristic, non-deterministic application of rules.

Developing computational design implementations requires interdisciplinary skills, which has been pointed by authors such as Fischer & Herr (2001) as one of the biggest difficulties in the teaching of generative design. In fact, good designers are not necessarily good programmers, and good programmers often cannot understand designers’ needs.

One possible reason for this incompatibility is the difficulty some designers have in dealing with symbolic reasoning. The fear that many architecture students have from mathematical subjects may be more than prose and could be related to a preference for visual figural cognition. It is not surprising that analogical computation is the principle of the established field of shape grammars.
Traditionally, figural cognition has been considered a more primitive type of thinking than symbolic reasoning, as stated by Guilford:

> At the preschool levels, one should not expect much in the way of visual symbolic abilities until after the age of six or much in the way of semantic abilities until after the age of two, but one might well expect to find some figural and behavioral abilities differentiated below the age of two. (Guilford, 1967, p.109)

Gardner (1999) claims that even Piaget focused on logical-mathematical – and therefore symbolical – intelligence instead of other types of thinking. One of Gardner’s intelligence categories, by contrast, is what he calls spatial intelligence and describes as “the potential to recognize and manipulate the patterns of wide space as well the patterns of more confined areas” (p.42).

According to Knight (1999-2000), in order to understand shape grammars and design computation in general, "a unique combination of technical, spatial, and intellectual abilities and interests is required" (p. http://www.mit.edu/~tknight/IJDC/frameset_issues_pedagogy.htm). However, she asserts that while computer implementations may "allow for rapid explorations of rules and design possibilities", they "may not be as effective as by-hand applications of grammars" (p. http://www.mit.edu/~tknight/IJDC/frameset_issues_computer.htm), fearing a "mindless" use of rules by simple trial-and-error, with superficial understanding of the process, especially by less-experienced, undergraduate students.

The present paper proposes a strategy for allowing these students to experiment with computational design and computer implementations at the same time, not by just giving them ready-made applications, but rather by teaching them an easy way to make their own design tools. The strategy is based on Papert's constructionist theories and Stiny's belief that trying to develop an algorithm for a given process is an excellent way to fully understand it (Stiny, 1978, p.208).

The following sections describe an experiment with what may be called analogical computer customisation and programming. The aim of this experiment was to develop strategies that could be easily learned by undergraduate architecture students, so they would be able to develop their own generative design tools. This undergoing research has started with the author’s Ph.D. thesis (Celani, 2002), and a couple of experimental courses taught in 2002. In 2003 a book with an introduction to AutoCAD VBA programming and exercises (Celani, 2003) was published in Portuguese, and a course called "CAD in the creative process" has been included as a mandatory subject for undergraduate architecture students at the School of Civil Engineering at UNICAMP.
2. Implementations

The basic idea behind the implementations that are described below consists of reproducing the steps that one would follow if developing a design computation with pencil and tracing paper. Two types of implementations have been tried, both using a popular “off-the-shelf” CAD application, AutoCAD (2000 or higher). The first type required no programming at all, but simply the customisation of the drafting environment. The second type involved Visual Basic for Application – VBA – programming, developed in AutoCAD’s built-in editor, VBAIDE.

The first type of implementation involved experimentations with symmetric and recursive designs. Arrangements of viewports with different zoom scales, rotations and reflections were created in Paper Space, involving overlapping and other non-orthodox computer-drafting methods. Commands such as Vpoint and Dview were used for respectively mirroring and rotating images inside the viewports. Figures 1 through 4 show examples of symmetric design environments and the respective AutoCAD commands used to develop them. After the environments were ready, students chose one viewport to develop a motif while simultaneously observing the resulting overall compositions, such as the ones shown in Fig. 5.

![Figure 1. Bilateral symmetry.](image)

In Model Space, draw any non-symmetric shape (such as the letter "R"). In Paper Space, create two viewports. Double-click on the right side viewport and type:

```
VPOINT
Enter
0,0,-1
Enter
```

Double-click one of the viewports to continue developing the motif design, while observing the dynamically updated reflection on the other viewport.
In Model Space, draw any non-symmetric shape (such as the letter "R"). In Paper Space, create three viewports. Double-click on the middle viewport and type:

```
DVIEW
Enter
All
Enter
TWIST
Enter
120
Enter

```

Double-click on the right side viewport and type:

```
DVIEW
Enter
All
Enter
TWIST
Enter
240
Enter
```

Double-click Paper Space and move viewports 2 and 3 on top of viewport 1. Double-click the overlapping viewports to continue developing the design. To obtain other cyclic compositions, use more viewports and the corresponding rotations ("twists").
In Model Space, draw any non-symmetric shape (such as the letter "R").

In Paper Space, create six viewports. Repeat instructions in Fig. 2 for the three first ones. Double-click the fourth viewport and type:

```
VPOINT
Enter
0,0,-1
Enter
```

Double-click on the fifth viewport and type:

```
VPOINT
Enter
0,0,-1
Enter
DVIEW
Enter
All
Enter
Enter
TWIST
Enter
120
Enter
Enter
```

Double-click on the sixth viewport and type:

```
VPOINT
Enter
0,0,-1
Enter
DVIEW
Enter
All
Enter
Enter
TWIST
Enter
240
Enter
Enter
```

Double-click Paper Space and move viewports 2 to 6 on top of viewport 1. Double-click the overlapping viewports to continue developing the design. To obtain other dihedral compositions, use more viewports and the corresponding rotations ("twists").
In Model Space, draw any non-symmetric shape (such as the letter "R"). In Paper Space, create many side-by-side viewports or a matrix of viewports. Double-click every other viewport and apply the rotation or the reflection techniques (or both) described above. Double-click the first viewport to continue developing the motif design.

The designs generated with the "viewport" strategy were compared to designs generated with traditional pencil and paper and computer drafting techniques (where a motif was first developed and then copied, mirrored and turned around). With the new techniques there were significantly more occurrences of emergent shapes and continuity between cells, due to the possibility of dynamically observing the final result while still developing the motif.

In a more “advanced” type of implementation VBA programming was used to develop parametric and algorithmic design applications. Instead of taking full advantage of the object-oriented language, a single VBA command was used to send messages to AutoCAD’s command prompt (ThisDrawing.sendcommand ("AutoCAD command")), combined with variable definition and code loops, in a VBA macro, a kind of script. It is
FROM SIMPLE TO COMPLEX

important to stress that the very same strategy can be used with AutoCAD’s other built-in programming language, Auto Lisp (with the “command” command), as well as with other programmable CAD packages. Figure 6 shows a step-by-step example of the definition of a rule-based composition and its corresponding script.

Figure 6. Simple ruled-based-design.

**AutoCAD procedure:**

In Model Space, draw any non-symmetric shape (such as the letter "R"). Create block "x" with this shape. Insert block "x" at point (0,0) with a given scale and rotation angle increment. Repeat this procedure \( n \) times.

**VBA macro:**

Insert a Module and add the following code:

```vba
'Block is the block's name
Dim Block As String
'SI is the scale increment
Dim SI As Double
'AI is the angle increment
Dim AI As Double
'i is an index number
Dim i As Integer
'NR is the number of repetitions
Dim NR As Integer

Private Sub Repeat()
    Block = "x"
    SI = 0.8
    AI = 10
    NR = 8
    i = 0
    Do While i < NR
        ThisDrawing.SendCommand (-insert & vbCrLf & Block & vbCrLf & "0,0" & vbCrLf & SI ^ i & vbCrLf & SI ^ i & vbCrLf & _
        i * AI & vbCrLf)
        i = i + 1
    Loop
End Sub
```
More sophisticated versions of the application above may include an interface and the possibility of defining the angle and scale increases and the number of repetitions wanted. Further customisation of AutoCAD and the use of some object-oriented techniques, such as the retrieval of object properties, can result in a complex application, like 3D Rules-Assistant (Figure 7), available for download at http://www.fec.unicamp.br/~celani/cad-criativo.htm.

Figure 7. 3D Rules-Assistant.

The techniques described above can be applied in education in many ways. For example, students can be given simple ready-made environments and applications to develop designs with, and then try to reproduce the code, based on the generated designs. Else, customisation and scripting techniques can be introduced, and then students may be asked to develop their own generative applications and to use them to develop new designs. The second option and the combination of both have been proven more effective so far.

4. Conclusions

Examples of rudimentary generative systems developed with simple customisation and VBA macros have been presented along with examples of their pedagogical applications. With these techniques, students who had no previous experience with programming were able to develop many interesting applications and designs.

The technique herein presented is not without limitations, and anyone adopting this strategy must be well aware of them. Not translating shapes into symbolic representations eliminates the possibilities of symbolic manipulations, which is the basis of evolutionary computation, just to cite an example. It also still lacks more experimentation with emergent shape detection. However, this technique can be successfully used as an introduction to generative design by computer without the risk of losing track of what is going on in the machine, once it allows students to start with very simple applications and progressively increase their complexity.
Acknowledgements

I would like to acknowledge professor Doris Kowaltowski, associate dean of the School of Civil Engineering of the State University of Campinas (UNICAMP), for her constant support and for providing me with many opportunities for testing my pedagogical methods at UNICAMP.

References

A STUDIO EXERCISE IN RULE BASED COMPOSITION

SOTIRIOS KOTSOPoulos AND HALDANE LIEW
Massachusetts Institute of Technology, Cambridge, MA, USA
skots@mit.edu | haldane@mit.edu

Abstract. This is the outline of a studio exercise that incorporates rule-based methods in design synthesis, using both analogue and digital means.

1. Introduction

One of the challenges in using a rule-based system is determining its appropriateness and applicability in the synthesis of form. How can rules be used to perform goal-driven design tasks? And how can analogue and digital rule-based systems coexist as part of the studio teaching activity? In order to examine these questions a studio exercise was developed on the basis of a design competition for low cost housing. The exercise aims to become a starting point for the introduction of rule-based methods, in synthesis.

The general strategy, given the building program, is to construct a method of producing a variety of 2-dimensional plan arrangements in response to a variety of functional demands and conditions. The objective is to gradually establish spatial elements, relationships, and rules for the generation of designs.

First, possible sets of spatial elements and rules are formulated with analogue means, as a hypothesis. Then, they are tested using a digital parametric shape grammar interpreter. The interpreter requires the conversion of the rules into a scripting format and provides computer aid in clarifying the ramifications of the hypothesis. Using the interpreter the designer determines if a rule-set produces any desired outcomes. If not, the rule-set is modified and re-tested. The decision process involves a selection among alternative rule-sets, where the designer explores the possible outcomes. The digital interpreter offers fast broad exploration of the products of the rules. The digital 2-d representations generated by the rules
could readily be used in solid modeling allowing alternative representations to participate in the evaluation.

The heuristics of the process are organized in three general, interdependent levels of abstraction. The first is dedicated to the formation of *partis*, the second to the transformation of a *parti*, and the third to the description of the tectonic details.

2. The Exercise

The exercise was based on a housing competition sponsored by the *Habitat For Humanity* (HFH) the summer of 2002 in Boston, Massachusetts. The HFH described the goal of the competition as: "the building of simple, decent, affordable houses". The program called for the design of adaptable types of 2, 3 and 4-bedroom houses without determining the square-footage of rooms or house types. All houses included: primary covered entrance, circulation, dining area, living area, at least one full bathroom, kitchen, and bedrooms.

![Figure 1. Examples of HFH housing in East Boston, Dorchester, and Roxbury](image)

A minimum living space limit for all house types was suggested: 900 s. f. for 2-bedroom apartments, 1050 s. f. for 3-bedroom apartments, and 1150 s. f. for 4-bedroom apartments. Further, the organizers did not designate specific sites, but offered several possible ones. Small, quadrilateral lots less than 5000 s. f. were an option, but lots larger than 20000 s. f. with complex shapes were also typical.
3. Design Concept and Method

The design approach is influenced by three factors: a) the absence of a designated site, b) the building program, and c) the provision for low construction-cost. The design concept is to develop rule-systems able to generate flexible house arrangements of variable size and morphology. The systematization of the ground plan is the method for the attainment of this objective. Analogue and digital means are both used in the design process.

The computational framework defined in Stiny 1980 within which shapes that belong in some algebra $U_i$ are composed with rules of the form $x \rightarrow y$, is employed in the production of design descriptions. Each design description is treated as a product of a finite device (grammar) that includes a finite number of rules and a finite vocabulary of spatial elements.

The proposed rule-based process generates descriptions in a “top-down” fashion. Similar models, referring to the construction of rule-based systems for 0-dimensional languages can be found in Carnap 1912, and Chomsky 1957. The possibility of establishing analogous methods in the analysis and synthesis of 2-d design descriptions was first discussed in Stiny and Mitchel 1978 in the production of Palladian villa plans. The grammar of Stiny and Mitchell captures the generation uniaxial Palladian villa plans in eight stages. Numerous papers have followed describing the generation of Frank Loyd Write’s prairie houses (Koning and Eizenberg 1981), Japanese tea-room designs (Knight 1981), Queen Ann houses (Flemming 1987), traditional Taiwanese houses (Chiou and Krishnamurti 1995), Yingzao fashi houses (Li 2000), and Alvaro Siza’s houses (Duarte 2001).

The novelty of the proposed approach is that it does not focus on an existent corpus of designs, but attempts to capture the exploratory effort of an intuitive creative process. A process of this kind involves selection among...
several candidate rule-sets, where the designer explores their possible outcomes using analogue and digital means. The proposed view is that first, the candidate sets of spatial elements and rules are formulated with analogue means, and then, they are tested digitally.

The heuristics of the process are organized in three general levels of abstraction. Each level contains finite sets of rules, the interaction of which is characterized by interdependence. At the top more abstract level of formation, the rules produce partis for possible designs. At the middle level of transformation, a specific parti is selected and transformed to a more detailed spatial arrangement. At the lower level of abstraction, that of refinement, the rules apply on the transformed arrangement to determine its tectonic details such as doors windows etc. All three levels of the process make use of analogue and digital means. The digital aid applies more drastically on the first two more abstract levels of formation and transformation.

The proposed framework can be described as follows:

$$\Sigma : \{ \text{finite set of spatial elements} \}$$

$$R: \{ \begin{align*}
\text{Formation} & : A_1 \rightarrow F_1 \\
& \vdots \\
& A_n \rightarrow F_n \\
\text{Transformation} & : G_1 \rightarrow M_1 \\
& \vdots \\
& G_k \rightarrow M_k \\
\text{Refinement} & : N_1 \rightarrow W_1 \\
& \vdots \\
& N_r \rightarrow W_r
\end{align*} \}$$

where $$A_1, \ldots, A_n, F_1, \ldots, F_n$$ are elements in $$\Sigma$$.

The analogue part of the exploration involves the articulation of candidate rules, while the digital part the rule-testing. Through an iterative process of formation, transformation and refinement similar to the see-move-see concept (Schon and Higgins 1992), the rules are evaluated and redefined according to their compliance to programmatic, intuitive, and construction criteria. Finally, the rules are organized in grammars.

A shape grammar digital interpreter (Liew 2003) is used for the digital part of the exercise. The interpreter, written in VisualLISP, uses a scripting language based on LISP to describe a rule. Each rule has four parts: left-hand schema, right-hand schema, transformation mapping, and variable mapping. A vector description format (Nagakura 1995) is used to describe the geometry and variables of a schema.

The transformation mapping determines any transformation changes between the left-hand schema and the right-hand schema. The variable mappings define a relationship between the parameters of both schemata. A schema is composed of two parts, the geometry and the constraints on the geometry variables.

The geometry of a schema is described using a series of vector displacements. Each vector has 3 components: action, vector and label. The
action component determines if the shape is a line or a point. The vector component describes the x and y displacement of the shape. The label component determines the name. For example, a horizontal parti line that is 5 units long is described as:

```lisp
((action "line") (vector 5 0) (label "parti"))
```

A shape is described as a series of vector displacements that are connected from end to end. For example, the following describes a “parti” square that is 5 units by 5 units in size.

```lisp
(((action "line") (vector 5 0) (label "parti"))
 ((action "line") (vector 0 5) (label "parti"))
 ((action "line") (vector -5 0) (label "parti"))
 ((action "line") (vector 0 -5) (label "parti")))
```

To describe a parametric shape, the numbers in the vector displacement description are substituted with variables. The following example describes a schema that finds all parti rectangles.

```lisp
(((action "line") (vector l 0) (label "parti"))
 ((action "line") (vector 0 w) (label "parti"))
 ((action "line") (vector (- l) 0) (label "parti"))
 ((action "line") (vector 0 (- w)) (label "parti")))
```

Restrictions can be set on the geometry variables to limit the type of sub-shapes found. These restrictions are added in the binding-constraints component of the schema. The following example restricts the size of the square to be less than 10 units.

```lisp
((binding-constraints
  (l (< l 10))
  (w (< w 10)))
```

To apply a rule of the form \( x \rightarrow x + y \), on rectangles

```
```

the program recursively searches the input shape for all instances of the left-hand schema and presents the possibilities to the user through an interactive menu that highlights the embedded schemata. Once the user selects an
embedded schema, the rule application is completed by subtracting the selected schema from the input shape and adding the right-hand schema of the rule.

The previous additive rule of the form \( x \rightarrow x + y \), applying on parti rectangles \( x \) and \( y \), can be expressed in the symbolic meta-language as follows (for simplicity, all the arrows are omitted from the shapes).

First, the left side of the rule,

```lisp
(setq schema-left-rule
  '((geometry
      ((action "line") (vector w 0) (label "parti"))
    ((action "line") (vector 0 h) (label "parti"))
    ((action "line") (vector (- w) 0) (label "parti"))
    ((action "line") (vector 0 (- h)) (label "parti"))
  )
  (parameter-constraints
    (w (> w 0))
    (h (> h w))
  ))
)
```

Second, the right side of the rule,

```lisp
(setq schema-right-rule
  '((geometry
      {action "line"} (vector w 0) (label "parti"))
    {action "line"} (vector 0 h) (label "parti"))
    {action "line"} (vector (- w) 0) (label "parti"))
    {action "line"} (vector 0 (- h)) (label "parti"))
)
```
Defining the transformation and parameter mapping,
(setq tmap-rule
  '((delta-xo . 0)
    (delta-yo . 0)
    (delta-ro . 0)
    (delta-za . 0))
)

(setq pmap-rule
  '((w w)
    (h h)
    (a w)
    (b (* 0.75 w))
)
)

And third, the connection of the left and the right sides of the rule,
(setq housing-rule
  '((left . schema-left-rule)
    (right . schema-right-rule)
    (tmap . tmap-rule)
    (pmap . pmap-rule)
    (success . nil)
    (failure . nil)
    (applymode . "single")
    (rulename . "housing-rule")
)
3. Results

How can rules be used to perform goal-driven design tasks in composition? The aim of rules in composition is to project a finite set of properties to a large set of compositions. But in order to identify the appropriate rules in synthesis, one needs to examine what they produce. Then, the rules can be organized to generate compositions with the desired properties. A grammar serves as a memory device: the rules are recorded and gradually classified with respect to the attainment of the objective at view. The grammar and the designs are the output of a broad discovery process, where spatial entities and rules are first distinguished and stated as a hypothesis, and then tested. They are transformed, and refined to achieve the purpose at view.

On what basis can we evaluate the products of rules? This is the practical problem regarding the use of rules in both design synthesis and the activities of the studio. In the case of grammars for speaking languages, a test of adequacy (Chomsky 1957) is to have the native speakers accept the produced sentences and to identify the false ones that are generated by this grammar. Chomsky assumes intuitive knowledge of the grammatical sentences of English, and asks “what sort of grammar is able to produce these sentences in some effective and illuminating way?”

In a similar fashion, in the analysis of a corpus of designs, the required initial properties and the rules that produce it, can be extracted from some original, previously analysed, instance (i.e. Palladian Villas, or Queen Ann houses, or Frank Loyd Write houses etc.) But, there is no predetermined criterion of evaluation in the synthesis of original compositions. The designer has to set the objective and the test at each step. The objective and the test remain open for re-evaluation as the testing proceeds.

In this study the choice of rules and designs is approached on the basis of programmatic-functional distinctions. Spatial relationships and rules that form different room-adjacencies are distinguished, and the values that determine the room sizes are gradually established. Large numbers of arrangements can be produced in this way. The need to develop more focused methods to control the generation of designs leads to the restriction of the rules.

How does analogue and digital rule-based systems coexist as part of the studio teaching activity? The spatial elements and rules are initially formulated with analogue means, because the ambiguity of analogue representation works in favour of the exploratory process. The absence of discreetness prohibits the early designation of values. The description
remains ambiguous, and yet useful for the time that questions of value remain uncertain.

The digital representation is more efficient in clarifying the ramifications of a candidate rule-set, by allowing the mechanical execution of large number of tests. In this way, the digital exploration helps to determine if a particular rule-set produces any desired results. If not, the rules can be modified and re-tested. The digital descriptions of rules require translation of the depicted shapes in symbolic form. Therefore, the values of variables within the rules need to be determined before the rules can take their digital form.

Sample descriptions from two working examples, at the three general levels of abstraction (formation, transformation and refinement) are exhibited in the next Figure 1,
Figure 1. Two working examples of designs, in 2-d and 3-d

4. Conclusions

In the proposed studio exercise the design descriptions are treated as products of a finite rule based device and generated in “top-down” fashion by a process that involves analogue and digital media. At the top level (formation), the rules produce parts. At the middle level (transformation), a chosen parti is transformed to a design. At the lower level (refinement), the rules determine the tectonics. Rule-sets are formed with analogue means and are tested digitally, to determine their outcomes.

A shape grammar digital interpreter is used for the purpose. The digital generative tool is particularly useful in the exploration of 2-d parts, at the stage of formation. Further, the transformation and refinement stages require the juxtaposition of several descriptive layers. For the juxtaposition of 2-d descriptions the digital tool used multiple Auto-CAD layers, and the symbolic expression of rules became increasingly complex. All the necessary 3-d descriptions were executed manually in Auto-CAD without using the interpreter.

Computations in 3-d that also require the juxtaposition of multiple descriptive layers will be the subject of future work.

References

Carnap R: 1937 (2002), The Logical Syntax of Language, Open Court, pp. 1-52
Flemming U: 1967, ‘More than the sum of parts: the grammar of Queen Anne houses’ Environment and Planning B: Planning and Design 14 323-350
Appendix

The following example shows the derivation of the formation and transformation samples of Figure 1 (p. 9). The formation includes the derivation of the 2-d parti, and the transformation the introduction of wall-layout, and openings. The example is part of the digital testing, where coded versions of the shape rules are used in the digital interpreter.

The example involves descriptions made out of lines that belong to three different Auto-CAD layers: Layer A includes only parti lines. Layer B includes only the wall-layout. Layer C includes only secondary shapes, of auxiliary character. The setting can be modeled in $<U_{12} \times U_{12} \times U_{12}>$ algebra. The three different layers are indicated by the (0, 0) cross-point of their coordinate system, and by the letters A, B and C, respectively.

The shape rule R1 generates a parti square in layer A. Layers B and C remain unchanged.

\[
\begin{array}{ccc}
   \mathcal{A} & \mathcal{B} & \mathcal{C} \\
   + & + & + \\
\end{array} 
\rightarrow 
\begin{array}{ccc}
   \mathcal{A} & \mathcal{B} & \mathcal{C} \\
   + & + & + \\
\end{array} 
\]

The rule R2 creates a parti rectangle from a given parti square in layer A, according to a specific proportion. Layers B and C remain unchanged.

\[
\begin{array}{ccc}
   \mathcal{A} & \mathcal{B} & \mathcal{C} \\
   + & + & + \\
\end{array} 
\rightarrow 
\begin{array}{ccc}
   \mathcal{A} & \mathcal{B} & \mathcal{C} \\
   + & + & + \\
\end{array} 
\]
The rule R3 also acts in layer A. The rule is of the form $x \rightarrow x + y$. It adds a parti rectangle to an existent parti rectangle. The length of the added rectangle is equal to the width of the existent rectangle. The Layers B and C remain unchanged.

The rule R4 also adds a parti square to an existent parti rectangle in layer A. Again, the side of the added square is equal to the width of the existent rectangle. The layers B and C remain unchanged.

The rule R5 adds a parti rectangle to an existent parti square, in layer A. The layers B and C remain unchanged.

The rule R6 adds a parti square to an existent parti rectangle, in layer A. The side of the added square is equal to the width of the existent rectangle. The layers B and C remain unchanged.
The rule \( R7 \) erases a \( parti \) line that lies inside an existent \( parti \) rectangle, in layer A. The layers B and C remain unchanged.

The formation of the \( parti \) is shown in the derivation,
An example of a *transformation* is exhibited next. The example shows the creation of a wall-layout. The presented rules follow the conventions of the digital interpreter. The following four shape rules are coded as one action, in the digital interpreter. That is, the interpreter executes all four rules each time the user picks the creation of a wall layout.

Rule R8 draws the wall-layout in layer B, in correspondence to a parti line in layer A. The existence of a *parti* in layer A is therefore necessary. The parti in layer A remains intact, and layer C remains unchanged.

\[
\begin{array}{ccc}
A & B & C \\
\hline
+ & + & +
\end{array}
\quad \rightarrow 
\begin{array}{ccc}
A & B & C \\
\hline
- & - & -
\end{array}
\]

The rule R9 executes the trimming of cross + intersections.

\[
\begin{array}{ccc}
A & B & C \\
\hline
+ & + & +
\end{array}
\quad \rightarrow 
\begin{array}{ccc}
A & B & C \\
\hline
+ & + & +
\end{array}
\]

The rule R10 cleans all T intersections.

\[
\begin{array}{ccc}
A & B & C \\
\hline
- & - & -
\end{array}
\quad \rightarrow 
\begin{array}{ccc}
A & B & C \\
\hline
- & - & -
\end{array}
\]

Finally, the rule R11 executes the trimming in L intersections.

\[
\begin{array}{ccc}
A & B & C \\
\hline
+ & + & +
\end{array}
\quad \rightarrow 
\begin{array}{ccc}
A & B & C \\
\hline
+ & + & +
\end{array}
\]
All four rules (R8, R9, R10, R11) are coded so that they are executed at once, one after the other. Therefore, in the derivation, the digital interpreter executes the transformation in one step.

At the end of the computation, the layer A includes the parti, layer B includes the wall layout, and layer C the empty shape.
Finally, in the next transformation two openings are created on the wall-layout, in layer B. For this purpose two rectangles are drawn manually in the third layer C of secondary lines. The rectangles indicate the positions of the openings. The addition of rectangles, in layer C, is expressed by the following shape rule R12,

\[
\begin{array}{ccc}
A & B & C \\
\hline
+ & + & + \\
\end{array} \quad \Rightarrow \quad \begin{array}{ccc}
A & B & C \\
\hline
- & & - \\
\end{array}
\]

The secondary rectangles are erased from layer C, during the application of the next rule R13, which creates openings in the wall layout (at layer B), and leaves the parti intact

\[
\begin{array}{ccc}
A & B & C \\
\hline
+ & + & + \\
\end{array} \quad \Rightarrow \quad \begin{array}{ccc}
A & B & C \\
\hline
+ & + & + \\
\end{array}
\]

The derivation,
At the end, the layer A includes the *parti*, layer B includes the transformed wall layout, and layer C the empty shape. The shapes in layers A and B are the *formation* and *transformation* samples of *Figure 1* (p. 9)
Abstract. Computational tools are becoming commonplace in both design thinking and making. While the nature of computation springs from an endeavor for generality and universal solutions, design seems to focus on a singular final result. The paper will present two software implementations for design: an application based on the principles of shape grammars, and a set of code-scripts developed for a specific architectural project. The possibilities and the difficulties in implementing computation-based systems for design purposes will be presented and conclusion will be offered for discussion.

1. Introduction

While design becomes more closely related with digital computation there are several novel effects emerging from this coexistence and parallel development. The subject matter studied and presented in this paper focuses on the essential difference in the nature of design being an act motivated and driven by a hunt for a singular end-product (Rowe, 1987); and computation as a science founded and influenced by principles of generalization and abstraction captured by the idea of a Universal Machine (Turing, 1963). Of course, none of these extreme modes of thinking are fully embodied in reality but nevertheless they tend to emerge and echo as opposing forces during the development of computational applications for design purposes. There is no implied conflict, though, but rather a chance for an inquiry on the interaction patterns of design and computation for the purpose of understanding the difficulties and potentials of creative applications.

Two research projects in design and computation will be presented which delineate the previously described thoughts. The first one explores the fundamental ideas of computation embodied in a shape grammars’ software and attempts to extrapolate its possible applications in design. The second one starts from a purely design-oriented investigation and on the way finds computation as the means for expanding the scope of the initial inquiry.

2. Universal Design.

sgStudio_min was an experimental general-purpose design software implemented for investigating the concepts of rule-based design. The paradigm of the Shape Grammars (Stiny and Gips, 1972) was used as a starting point. The term rule-based characterizes a process described as a succession of decision-steps. In a greater context these processes originate from the theory of
the Formal Languages. A language according to this theory is defined as an arbitrary set of words; words are finite sequences of letters / symbols; sets of these symbols constitute alphabets; generative grammars are defined by a finite alphabet of non-terminal symbols, another finite alphabet of terminal symbols, a starting symbol from the first alphabet, and a list of ordered symbol pairs; finally, a generative grammar produces a language (Révész, 1983). Formal languages have been used in many fields other than mathematics and computer science. For instance, Noam Chomsky’s theory on languages (Chomsky, 1957) was closely related to this field. Furthermore, the theory of Automata was encapsulated by formal languages. It is fair to say that Shape Grammars departed from this theory based on shape manipulations rather than symbols (Gips, 1975; Stiny 1975).

A rule, in this context, describes a relationship between shapes; the relationship is further defined as a two-state condition: an initial configuration of space and a final one. The concept of rule-based derivation is quite simple: “under these conditions, take the following action” or “if this spatial configuration is found, then change / modify / replace it with this one”. A rule therefore, describes a conditional execution of a design action. Ultimately, a design process is expressed globally as a sequence of rule application.

The conceptual framework of the application was founded on the idea of “no observer elimination”. The idea was that the observer/designer is the most crucial part of an application developed for design purposes and a substitution or elimination by a computational observer or a design machine/automaton was to be avoided. I begun the design of the application by first observing how people actually derive spatial configurations by following basic shape-grammars’ rules on wood-made toy blocks. The cognitive problem of deriving a shape grammar is found on the geometrical complexity that is piled up sequentially in each step. Each basic transformation has a different degree of difficulty, as for example it is easy to apply translations, more difficult to apply rotations, extremely difficult to resolve reflections and impossible to apply scaling given the inflexibility of the physical medium. In order to cope with these perceptual difficulties there were some mental-helpers employed such the re-entry or re-initialization behavior. In the first case, I observed that while applying rules people are constantly grouping and regrouping the shapes that have been already placed in the configuration in order to find a pattern that would allow the originating rules to be grouped in a recursive sub-process and thus discover a re-entry point of a loop. In the later case, I noticed that in order to proceed to a next step of a derivation, based on a depicted rule, people usually attempt to simplify the transformation application by trying to transforming the whole configuration until the application shape(s) matches the depicted rule pair’s left-hand space.

These behaviors were extremely interesting and the software was mainly traced upon them. sgStudio_min was implemented in Java <http://java.sun.com>, which is one of the most highly developed computer languages. The specific encoding medium of object oriented programming accommodated the process of translating the observed behaviors in a computational format. For instance, each shape was defined by a computational geometry and augmented by a polymorphic list of transformations. The idea of a global re-initialization was interpreted by the means of bipolar inheritance of transformations, where a shape applied in design under a rule, inherited all the transformations of the shape it replaced in the target space and all the transformations defined in its right-hand side of the rule. Finally, the most problematic aspect of shape grammars of pattern matching and indifference of transformations in rule application was handled by a layered system of conditional tests: class type matching, parameter matching, transformation list matching and finally geometrical matching.
The final application was rather closer to the symbolic model of Formal Languages than the idea of Shape Grammars, which is based on a non-symbolic definition for geometry (Stiny, 2000). sgStudion_min can be described as a geometrical infix interpreter/evaluator or a random-access machine that uses a stretchable storage, in which symbolic shapes can be inserted in a montage-like fashion. As it becomes evident, the random access characteristics are inherited from the encoding medium, the digital computer and the programming language, and the infix characteristics are the results of the attempt to encode a human-like behavior by these means.

Figure 1. sgStudio_min version 1.0.

The first version of the software offered a great amount of intuitions regarding the specific computational formalization of the design process (Figure 1). Initially, it became apparent that the procedural framework, the rule-based approaches offer, is extremely generative powerful. Numerous variations of highly complex configurations can be produced by simple sets of rules describing basic spatial relationships. Therefore, the generative potential of the model is extremely high. The generative model captured by this application though, was bounded by a single-directional rationale; begin with almost nothing and produce almost everything by building, upwards or downwards, sequentially more complex structures based on basic elements and methods. Design as a process, cannot be adequately simulated with such model, because it also involves multiple other tactics. Abstract design intentions cannot always be structured in this continuous sequential mode, but rather appear or evolve sporadically in a punctuated equilibrium fashion. Furthermore, design cannot be entirely either global or local, top-down or bottom-up. The fact that eventually can be presented linear as such, does not justify the assertion that it is essentially linear.

The evaluation of the application, from this perspective, is that it would be highly cumbersome to develop a design with some level of sophistication based on these principles. The problems were mainly found on the cognitive level: it is very difficult to retain control in building a very complex design product based on simple geometrical manipulations. After a number of defined rules, the process becomes rather one of logistics, because it is impossible to keep track of all components. There was also a problem of scalability, because the attachment to a graphical-shape notation is rather constraining from developing intuitions and abstractions for the design. Finally, the whole model was so highly syntactic, that made difficult the process of evoking or attaching a meaning in a designed object.

The reflections on rule-based design, formulated the basis for the second version of the software application in which I attempted to expand or divert the scope of the investigation in creating a geometric-design programming language. The idea for such a project emerged from
both the feedback gathered by the initial implementation of the software and an attempt to further evaluate the potential of rule-based system.

A programming language is defined by only a few basic linguistic constructs: information storage and access, iterative execution and conditional execution. Higher level constructs consequent of the initial set are: nested execution and recursive execution. A programming language thus is a computational generative model. Realizing that the rule-based application covered all of these aspects, it was quite inevitable to move in that direction. A rule encapsulates a conditional execution and furthermore it contains a description of design information. Gradually the act of design by sequencing rules started becoming more like a computer programming practice.

![Figure 2. sgStudio_min version 2.0 alpha.](image)

The objective for the new version of the software was to capture the essence of this observation in a rather tangible way (Figure 2). The graphical user interface was composed out of four design control panels. Two of them represented the two parts of a rule: the initial space and the eventual space. A third panel contained the geometry of the design canvas: a space where the results of the rule application would be ideally visualized. Finally, the last panel contained a diagrammatic description of the defined rules, the contained objects in the rules’ spaces, the properties of these objects and the sequence of rule-based commands. The user would primarily interact with the diagrammatic interface entitled as the linker panel. Through this interface it would be possible to create definitions of objects and their transformations, connect their common properties, define interactive rules and finally create chains of design commands. One of the most powerful features was encapsulated in the idea of rule-space connection. This feature would allow rules to interact with each other by linking their spaces (initial and eventual). Through this mechanism it would be possible to generate non-linear relationships in a cognitive-friendly manner. By the same means also, the initial abstraction of a deterministic tree or lattice diagram would be extremely expanded.

The project reached a level of development where it became obviously more useful as means for explaining the basics of computation to designers by employing the means of geometry, rather
than a tool for actually designing. Of course it is difficult to evaluate a software application developed to be general-purpose in an empty design context. This was actually the most vivid lesson learned from this project: design tools are better off co-developed along a design intention than in blank space. The idea of rule-based system then becomes valuable as means of assembling geometric machines for testing design ideas. A shape-interpreter might not be an ultimate-solid-universal design machine but rather a sparse-flexible-procedural toolkit for experimentations. The next project illustrates this point by taking the encoded legacy of this project into a more integrated process of design and computation.

3. Case Driven Computation

The MiranScript project involved the implementation of a generative algorithm for prototyping and fabrication purposes of the existing gallery design. In a greater scope, this project was an in-depth continuation of a previous research in rapid prototyping and digital fabrication. Instead of exploring the generic possibilities of computational methods for design, the MiranScript project focused on the descriptive generation of prototyping and fabrication information for a single case scenario design.

The initial goal was to automate a process of fabrication based on the decomposition of a curved form into planar elements by means of contouring. This approach is commonly used for translating complex three-dimensional objects into sets of two-dimensional components. Contour-slicing makes possible the realization of complex geometries by describing them as simpler elements that can be fabricated by current CNC technologies. The obvious advantages of decomposing a complex object into flat pieces come with a number of limitations springing from the generic nature of process. In other words, the process of contour generation is on the one hand abstract enough to process every kind of operand geometry, but on the other, this abstraction collides with the specificities of a preconceived design solution. Therefore, it can be said that there is a general problem concerning generic processes and design. Generic processes are unable to adapt to the conditions of specific design intentions. Simultaneously, generic processes tend to homogenize the outputs which have to be differentiated afterwards. The effects of the homogeneity, which are highly visible in digital design projects, are caused by the utilization of arbitrary computational processes as this. This observation hints about the necessity of a co-development of both design intention and the computational expressive means.

During scripting of the contouring function it became obvious that the schema of the process could be expanded. A contouring method is controlled by an axis along which perpendicular planes are defined and slice the operand geometry. The intersection produces curved profiles used later on for creating extruded rib members. Usually, the stepping function from each plane to the next one is constant. Contouring of this fashion is usually employed by architects and planners for representing terrains and landscapes. The concept of the contouring script was based on the idea of polymorphism. Polymorphism, characterizes behaviors of objects unrelated to their underlying description and parameters. For instance, the contour slicing is defined by an axis and a stepping function. There is no inherent constraint for using a line as this axis and a constant pace as the stepping function. Any method based on the essence of slicing incrementally along an axis can be thought of as a contouring mechanism. Moreover, slicing does not intrinsically imply cutting a curved object with a plane, but rather a generic intersection operation between any number of objects, of any type as long as they are greater than one.
The scripted process used a spatial curve for an axis and a non-uniform stepping function for controlling the density of slicing (Figure 3). On each point on the curve, the tangent vector was used to define a plane which then sliced the original model. The density of the point distribution on the curve was controlled by an external diagrammatic operator. Therefore the stepping function was described geometrically by the user and then employed by the script in order to vary the density of the slicing process. According to a control parameter, the generated cutting plane could either intersect the model and generate a single profile or offset itself in both normal directions, by a user-defined amount, and then slice the model twice. In the first case the single profile was used for generating extruded rib members and on the second for creating rule-based curved ribs. Therefore, there was a conditional execution pattern introduced in the process. In the first instance the ribs could be fabricated by most current machining techniques, such as laser-cutting, but on the second one, because of the curved side of the members they could only be manufactured by specialized machines with multiple axes of motion freedom. The same script, under slightly different parameters also generated a slotted base structure for the ribs to be positioned and locked in place on the floor. Finally, a last scripted procedure generated joints between the ribs by following the iso-parametric curves of the original surface.

The results of this generalization created immediately a variety of advantages over the simplistic contouring technique but also new issues concerning the control of such multi-variable processes. The introduction of spatial curve as an axis of slicing and the ability to vary the density of the ribs augmented the process by allowing a gesture to be embedded in the abstract geometry. In other words, the arbitrary contouring mechanism, which disregards all intrinsic characteristics of the design and treats all input geometric entities equally, became a relatively more expressive mechanism. The expressive potential came both from the objective characteristics of the geometry which were now was actively considered through the process and from the subjective fact that the final outcome exhibited formal dynamic properties. Therefore, the curvature of the axis, variance of the density and the iso-parametric jointing densified the characteristics of the product and simultaneously embedded an intrinsic tension on the form by operating on its sub-components globally.
The script initially requested the density parameter to be encoded as a parametric mathematical function. For instance, a constant density could be described by a single number (the pacing distance), a linear increase could be encoded as a first degree expression and for more complex behaviors elaborate expressions, utilizing the underlying scripting engine, could be used. This approach became cumbersome because sophisticated mathematical description demanded non-design-friendly knowledge-base on math and also because the model of hard-coding information in this symbolic manner is highly non-intuitive. For these reasons, I decided to change the representation from symbolic to geometric and allow the user to draw a graph by means of the geometric tools of the CAD application. The script interpreted the graph-curve and extracted the implicit density information. The representation shift was decisive because of the visual quality of the graphed mathematical expression. Scientifically, there was a relative loss of control and precision over the process because the NURBS polynomials cannot describe all mathematical curves, but nevertheless the usability increased dramatically. Yet, the graph, even its initial cognitive advantages, it remained an abstract metaphor from the intuitive idea of controlling the density of points distributed along a curve. After a long struggle to embody this design information I ended up hacking the problem by introducing a well-known interface for controlling distributions used in most photo-editing software. The density curve, which was eventually the parameter the script requested, was inspired from the tone-curve used for lightness-contrast calibrations in digital imaging (Figure 4). This simplistic interface has a single extremely important property; it is considered as common knowledge to all designers using computers.

The experiment became also interesting in terms of scalability and control because the process was effectively depended on multiple parameters of various sources. Some of the parameters were numeric and other geometric. The design of a controlling mechanism was mandatory since the process would have been impossible to be employed in meaningful ways otherwise. This experimentation was highly important since most processes as such, reaching a critical mass of complexity, cannot be essentially controlled by simplistic parameters. The process in its entirety described previously defines a mechanism which has specific inputs, outputs, internal behaviors and external parameters. These kinds of processes are in a sense primitive. They operate directly on top of geometry and generate or filter design information. It is laborious yet relatively trivial to define by scripting or parametric modeling such multi-variant systems. The only constraints in these cases are the level of understanding of how computational geometry functions and the level of clarity of what has to be described by such means.

Design processes usually expand fast in breadth because there are multiple equally important conditions that have to be considered simultaneously, such as form, function, structure and materiality. Design also evolves equally fast in depth because each of these conditions are refined
and new considerations have to be handled in each level of detail. On this map of interrelated and expanding processes of design, a single computational mechanism is rather difficult to encapsulate the whole phenomenon. The Miran gallery script exemplified the idea of the evolutionary singularity of a computational design mechanism: initially by expanding an architectural idea through parametric encoding and allowing controlled exploration of its potential; and finally by focusing and handling the design specificities that sprung from the initial explorative phase. The further the process moved deep into design details the more the script became less general and more architectural. Eventually, there was one script tailored for the project: the MiranScript.

4. Conclusion

Both of the presented projects were designed and implemented as means of exploring the potential of computation for design purposes. The common thread along which both were developed inquired for a pivotal point between design and its tendency to focus in a final singular product in relation with computation which is usually driven by principles of generalization and abstraction. sgStudio_min illustrated a process of abstracting an ideal/virtual design in a purely procedural manner, described by geometric rules and relationships, and the underlying intent to produce a universal-flavoured platform for its birth and development. In this process of abstraction possibilities of design were actually narrowed down rather than expanded, because of the imbalanced relationship between an unbounded endeavour of reduction of design in its constituent elements—with the ultimate goal of its later reconstruction—and a design which may be procedural but nevertheless driven by the desire of appreciation of its singular end product.

On the other hand, the script written for the Miran gallery initiated an inquiry on computational applications for design from the rather modest, narrow in expectations, technical-flavoured context. During the design process and the concurrent development of its computational solution, the scope of the initial inquiry was expanded by understanding that what was eventually fabricated was not merely an automation system. On the contrary, MiranScript became an expressive medium; a generative system that did not target to encapsulate the whole design process from start to the end, but to strategically embed itself in the process and accommodate the design in order to evolve and form a unique identity.

Through this comparative investigation it became evident that a co-evolutionary scheme between the means and the ends tends to operate to a certain extent in favour of both design and computation. The difficulties illuminated by these projects, such as complexity and scalability, need further development in terms of abstractions that will allow a merge to be possible. A future development in this stream of investigation will have to focus in further clarifying the potential of blending higher level of design considerations, such as human and cultural factors, with computational interpretations and explore the possibilities of their integration.

Acknowledgements

sgStudio_min was developed in the Computation Design I: Theory and Applications fall class of 2002 offered by Terry Knight (Associate Professor of Design and Computation, MIT).

The Miran gallery script was part of the Architectural Design Workshop: The 2003 Centre Pompidou Exhibition Workshop offered by Mark Goulthorpe (Associate Professor of Design, MIT). This project would not have been possible without the invaluable help of my friend Alexandros Tsamis (SMArchS Candidate of Design and Building Technology, MIT) who guided the process of establishing a design sensitive and intuitive user input/interface experience, with his insightful comments and recommendations.
References

Stiny, G: 2000, How to calculate with shapes, Cambridge, MIT.
GENERATIVE DESIGN: FROM ALGORITHMS TO INTUITIVE AND INTERACTIVE SOFTWARE DEMONSTRATORS

KRISTINA SHEA AND MARINA GOURTOVAIA
Cambridge University Engineering Department
Engineering Design Centre (EDC)
Trumpington St.
Cambridge, CB2 1PZ, UK
ks273@cam.ac.uk, mg326@eng.cam.ac.uk

Abstract. This paper provides a brief presentation of eifForm, a software demonstrator for generative structural design and optimization based on a method called Structural Topology and Shape Annealing (STSA). The eifForm project aims to create and maintain a useable and stable prototype system that best demonstrates the research to date, allows for modular integration of new research advances and enables important feedback to be gained from potential users throughout method development.

1. Introduction

Performance-based generative design and optimization methods aim to create new, collaborative relationships between designers and computers to expand designers current capabilities. Software demonstrators that embody these algorithms in an interactive, flexible, and simple way hold great promise for making the research methods more appropriate and accessible by enabling important feedback to be gained throughout method development by the researchers themselves and from potential users. Presently, the specific nature of computer-aided engineering (CAE) systems has been recognized (Raphael and Smith, 2003), a solid body of research into Engineer-Computer Interaction (ECI) has been published (Parmee and Smith, 2002), and research into models of interaction for generative systems in architecture has been carried out (Chase, 2002).

Several generative systems have been implemented that enable designers to better explore unique and complex spatial compositions. Based on work in shape grammar formalisms by Stiny (1980), a general system for generating architectural massings was developed (Wang and Duarte, 2002) as well as a more focused system for generating spatial constructs that mirror musical compositions (Economou, 2000). Recently, Duarte (2004) has developed and implemented a grammar for the generation of mass customized housing designed by the architect Álvaro Siza at Malagueira in
Portugal. Using evolutionary processes, a system for free-form surface generation, called MOSS, has been implemented based on Lindenmayer systems (Testa et al., 2000) as well as a system called Agency GP, based on genetic programming, for interactive design exploration in a 3D modeling environment (O'Reilly et al., 2001). A recent system called Genr8 (Hemberg et al., 2004) has also been developed for surface generation using a grammar-based generative growth model. Genr8 has been implemented as a plug-in for Maya.

Generative design software demonstrators that move beyond spatial generation alone, to include performance models, analysis routines, and optimization algorithms, have been slow to emerge. One system based on an evolutionary algorithm has been developed that includes performance feedback from lighting and thermal analysis to generate novel and performance driven window and shading systems as well as roof geometry (Caldas, 2002).

To achieve usability of generative and optimization methods for both structural designers and architects, rapid prototyping of software applications is essential in order to demonstrate potential impact for design practice and enable feedback throughout method development. To create a robust yet flexible platform for the development of computational design demonstrators, a Computer Aided Engineering Design Research Methodology (CAEDR) that supports rapid creation and evaluation of next generation computer aided engineering tools has been proposed (Bracewell et al., 2001). From this research, a software development platform for creating eifForm, a generative structural design and optimization tool, was defined (Bracewell and Shea, 2001). The eifForm project has delivered a usable demonstrator that is attracting attention from practicing designers and architects. This project has also provided valuable insight into the way such demonstrators can be designed and implemented.

2. Structural Topology and Shape Annealing (STSA)

eifForm is the software demonstrator for the generative method called structural topology and shape annealing (STSA). This method combines structural grammars, performance metrics, structural analysis, and stochastic optimization (Figure 1). STSA supports exploration of discrete structural forms in relation to engineering and architectural performance for both routine and challenging scenarios. Compared to published results, the method is capable of generating multiple design alternatives for planar truss, single-layer spatial truss and full-scale transmission tower design tasks that are innovative yet efficient (Papalambros and Shea, 2002). More recent advances have included development of a new structural grammar for 3D truss-beam structures, including new automatic structural analysis
techniques, that has been used to design a light, slender cantilever structure for a novel noon mark installation, which has recently been built in London (Shea and Zhao, 2004).

The core methods of the software (Figure 1) are broken down into four main modules: grammatical design transformation, e.g. structural grammars, structural analysis, performance evaluation, and stochastic optimization, e.g. simulated annealing optimization. The generation of structural designs is performed by iteratively applying grammatical transformation rules, chosen from a pre-defined set, to an initial design. The input to the core algorithms expresses design intent and consists of a geometric description of an initial design, specifications that constrain grammatical transformation and define the structural model, and performance requirements and design goals expressed in the form of design constraints and objectives. For further information on the method see (Papalambros and Shea, 2002).

![Figure 1. Structural topology and shape annealing (STSA) method overview](image)

### 3. eifForm

While theoretical comparisons to structural topology optimization benchmarks have validated the STSA method, evaluation of its potential for expanding designers capabilities is only made possible by enveloping the core algorithms into an interactive shell with a simple graphical user interface (Figure 2) that highlights the unique attributes of the approach. To meet this aim, specification requirements for the eifForm project that are related to the user include:

- providing a simple, intuitive GUI with low latency that allows easy accessibility to potential users with different backgrounds, e.g. architects and engineers;
transparent description of input models (Figure 1) consisting of design context,
performance requirements, and generative parameters;
interactive selection and application of available structural grammar rules to
foster better understanding of their capabilities and effects (Figure 3);
regular visual updates of the design model during automatic design generation
via the optimization process;
performance feedback, including structural properties, loading conditions and
qualitative analysis results, displayed in the GUI for better understanding and
interpretation of the structures generated;
allowing interruption of the optimization process at any point;
fast execution of structural analysis and optimization processes;
archiving and providing access to the large numbers of design alternatives that
are generated and their input models;
providing means for integrating eifForm with other computer tools commonly
used in design, e.g. CAD and analysis formats.
Additional software specifications related to software development include:
fast, low-cost development of robust, easy to maintain software;
straightforward integration of research advances into the demonstrator;
dual-platform (Linux and Windows) development.
All points in the software specification have been initially addressed. Selected software solutions that have been implemented to ensure compliance with these requirements include:

- use of XML as a transparent means of describing input models;
- achieving fast real-time rendering of design models as they evolve in 3D with OpenGL, a platform-independent API for 3D imaging;
- maintaining a high speed of algorithm execution in the presence of a GUI with enhanced interactive features and real-time 3D rendering of both geometric and structural models by giving the user an option to define the user’s level of interaction with the computational process;
- using multi-threading to provide low latency user interaction for a computationally-intensive processes;
- designing a flexible system that is characterized by low cohesion between highly specialized modules and allows straightforward integration of research advances as plug-in modules, for example, new optimization and learning methods or a grammar rule interpreter (future work);
- supplying the scripting shell with modules for reading in and writing out information about designs in different standard formats (XML, VRML, DXF, FEIt) thus creating interfaces with other CAE tools (CAD, structural analysis packages, etc.) that either precede or follow the use of eifForm.

Further details can be found in Gourtovaia et al. (2004).

The current demonstrator provides access to the structural grammars for planar trusses and single-layer spatial truss enclosures, e.g. domes. Further structural grammars have been implemented and benchmarked, i.e. 3D truss-beams, structural systems composed of truss-beams, and transmission towers, and their incorporation within eifForm is underway. Improvements to all aspects of the GUI to better meet the goals of the specification are also under continuous development.

The software is implemented using a two-language approach (Bracewell and Shea, 2001), with performance-critical core modules written in a compiled language, ANSI C, and the main control loop, GUI and all supplementary modules written in the Python scripting language, which is advantageous for customization, e.g. incorporating new performance models. This development approach has enabled a dual-platform (Windows and Linux) software development environment that facilitates implementation of stable prototypes and fast incorporation of research advances.
A software demonstrator provides a means for gathering related research advances into a single tool, gaining insight about method functionality and potential applications as well as disseminating the research ideas to potential users, both students and practitioners alike. To date, eifForm has been used in uncontrolled experiment in collaboration with several architectural studios at MIT and the Architectural Association in London. In addition, a few practicing architects have been exploring eifForm. However, use of the system in practice for full-scale design projects often requires eifForm to be customized and driven by the authors. It is intended that once a design scenario is set-up within eifForm and the required customizations to the method and demonstrator have been made, that it can then be given to a designer to carry out their own design explorations. The current strategy to increase use of STSA in practice and its potential for impact is to carry out full-scale design projects in collaboration with both architectural and engineering firms to develop a series of practical case studies.

4. Conclusion
Interactive computational design applications grow out of theoretically proven algorithms for computational design. However, high-quality
algorithms are just one necessary pre-requisite for potential effective use in design practice. Evaluation of generative design and optimization methods is only possible through the creation of software demonstrators. Taking a modular two-language approach, creating an intuitive, interactive interface and maintaining the speed of both the core methods and the GUI has resulted in creation of a software demonstrator that successfully illustrates the research advances.

Acknowledgements

Current research support for this project is provided by The Leverhulme Trust (UK) and an IMRC grant from the Engineering and Physical Sciences Research Council (UK).

References


EMERGENT DESIGNER

An Integrated Research and Design Support Tool Based on Models of Complex Systems

RAFAL KICINGER, TOMASZ ARCISZEWSKI
CEIE Department, George Mason University, Fairfax, VA 22030 USA

and

KENNETH DE JONG
CS Department, George Mason University, Fairfax, VA 22030 USA

Abstract. The paper introduces an integrated research and design support tool, called Emergent Designer, developed at George Mason University. It is a tool that implements models of various complex systems, including cellular automata and evolutionary algorithms, to represent engineering systems and design processes. The system is intended for conducting design experiments in the area of structural design and for the analysis of their results. It implements state-of-the-art representations supporting generation of novel design concepts and efficient mechanisms for their subsequent optimization at the topology and sizing level. It also implements advanced methods, models, and tools from statistics and from the linear as well as nonlinear time series analysis to conduct the analysis of the design processes. Thus, it is a versatile tool that can be used both as a state-of-the-art design support tool and as an advanced research tool equipped with the methods and tools for the analysis of the design processes and of the obtained experimental results.

1. Introduction

With the emergence of Information Technology, new design methods are being developed which are based on various computational models of design processes. However, up until very recently, computers in structural design were used mostly and merely for various analytical design activities conducted in the final part of the engineering design process, namely in the detailed design stage (Arciszewski and De Jong 2001). Today, we are finally witnessing the emergence of new design methods applicable both in
the conceptual and detailed design stages. In order to fully benefit from this progress, these emerging design methods require new computer tools.

Another motivation comes from the fact that there is a growing trend to apply evolutionary design methods not only to strictly optimization tasks but to use them in finding creative/novel design concepts. Representations of engineering systems are one of the key issues to achieve this goal. When the focus is on finding an optimal design, the attention is usually restricted to a particular concept or at most several concepts of existing designs. In this case, design representations usually take a form of parameterizations of an engineering system, or its parts. Traditional representations frequently used in engineering optimizations problems, like binary representations, integer representations, real-valued representations can be included in this category.

Creative evolutionary design requires, however, more general and usually more complex representations. Representations that have been used in creative design are diverse but nevertheless share some similarities. Typically, phenotype representations are quite general and thus capable of representing large numbers of alternative shapes, forms, or morphologies (Bentley 1999). They range from direct representations, as in voxel-based representations (Baron et al. 1997) or array-based representations (Kane and Schoenauer 1995), to highly indirect representations, i.e. representations that do not encode solutions but rather rules on how to build these solutions. The most popular examples of indirect representations are grammars (Roston 1994), trees (Funes and Pollack 1999), shape grammars (Stiny 1980), cellular automata (Frazer 1995), L-systems (O'Reilly et al. 2000), and embryogenies (Bentley and Kumar 1999).

In this paper we introduce Emergent Designer, a computer tool developed at George Mason University, that addresses both important issues mentioned above. First, it implements a new design method called Emergent Engineering Design (Kicinger 2004) which is applicable both in the conceptual and detailed design stages. Second, it emphasizes both important objectives of the structural design process, i.e. development of novel/creative designs and their subsequent numerical optimization. The development of novel/creative designs is supported by indirect, or generative, representations based on various models of complex systems.

A general overview of the system describing its overall architecture and the flow of information is presented in the following section. Section 3 offers detailed description of its all major components. The actual implementation of Emergent Designer is presented in section 4. Finally, section 5 presents conclusions and recommendations for further work.
2. System Overview

*Emergent Designer* is intended for conducting design experiments in the area of structural design and for analysis of their results using methods, models, and tools from statistics and linear as well as nonlinear time series analysis. Thus, it can be used as a design support tool equipped with the state-of-the-art mechanisms for the generation of creative/novel design concepts and for conducting their optimization. Moreover, it is at the same time a versatile research tool that implements advanced methods and tools for the analysis of the design processes and of the obtained experimental results.

The following subsection provides an overview of the architecture of the system and briefly describes its major components/modules. Subsection 2.2 presents the information flow diagram and discusses integration issues as well as interactions among the components of the system.

2.1. ARCHITECTURE

*Emergent Designer* consists of several components. They can be divided into three major groups:

- **Design components**
  These components implement Emergent Engineering Design (Kicinger 2004), the design method which uses models of complex systems to represent structural engineering systems and design processes. They form the core of the system and implement the actual design processes.

- **Analysis components**
  These components implement tools and methods for the analysis of the experimental results and design processes. The components included in this group are aimed to provide quantitative information about the conducted design processes as well as statistical estimates of the performance of the design method. They are also intended to provide deeper understanding of the dynamics of design processes and the structure of the design spaces from global/holistic perspective.

- **Visualization components**
  These components implement various visualization methods and report generation mechanisms. They contain tools for visualization of the results of various analyses, e.g. statistical or time series, conducted by the system’s components. Also, automated tools of generating experimental reports that include detailed descriptions of experimental parameter settings and obtained results are implemented.
The high-level overview of the architecture of *Emergent Designer* is presented in Figure 1. It shows the individual components of the system contained in each of the groups discussed above.

![Architecture of Emergent Designer](image)

2.2. INFORMATION FLOW

The flow of information in *Emergent Designer* is presented in Figure 2. It provides an overview of the relationships among the components discussed in the previous section and shows where user input/decisions are expected. It doesn’t show, however, the information flow within the individual components. They are discussed individually in section 3.

Once *Emergent Designer* has been started, the user has the choice of conducting a new design experiment or using advanced statistical and time series analysis tools to analyze experimental data from the previous experiments. By default, a new design experiment is selected and the **Problem Definition Component** is called to define a design problem.

**Problem Definition Component** is intended to select a domain of interest, e.g. steel skeleton structures in tall buildings, and a specific design problem that will be solved, e.g. design of a wind bracing system. The component allows for the specification of values of the parameters defining the considered design problem, e.g. the number of stories in a tall building, or the height of a story. **Problem Definition Component** also implements mechanisms for saving the system’s parameters and their values to a file, and retrieving previously saved values from a file.

When the design problem is completely defined, the user has to decide whether, or not, to decompose the problem into several sub-problems. **Representation and Decomposition Component** is used for this purpose. If the design problem is decomposed, then one of the several decomposed representations can be selected. On the other hand, if the design problem is not divided into sub-problems, then the spectrum of possible representations
EMERGENT DESIGNER

includes the parameterized representations, the generative representations, and the self-adaptive generative representations (see section 3.1 for more details). *Representation and Decomposition Component* allows for the specification of values of representation specific parameters, e.g. shape of the local neighborhood in a cellular automaton used in the generative representations.

\[\text{Figure 2. Information flow in Emergent Designer}\]
When the design problem and its representation have been defined, the **Concept Generation and Optimization Component** is used to specify what type of concept generation mechanism will be used and whether, or not, topology optimization and/or sizing optimization mechanisms should be employed. If only pure concept generation mechanisms are selected, i.e. no optimization, then design concepts will be generated by the process of iteration conducted by cellular automata. On the other hand, if evolutionary algorithms are employed, then either only optimization of the design concepts is performed (when parameterized representations are used), or the generative representations are combined with optimization mechanisms to produce the design concepts.

The produced design concepts are transferred to the **Evaluation and Simulation Component** which evaluates them and assigns fitness value(s) (multiple fitness measures are used in the multiobjective optimization). This component is used to select the evaluation model assumed in a given design experiment and the values of evaluation specific parameters, e.g. methods of determining wind loads acting on the steel structure, or magnitudes of dead and live loads, etc. Also, simulation parameters, including the number of runs, the number of fitness evaluations, have to be defined in order to run a design experiment.

The four components described earlier, i.e. **Problem Definition Component**, **Representation and Decomposition Component**, **Concept Generation and Optimization Component**, and **Evaluation and Simulation Component**, form a group of design components that implements Emergent Engineering Design (Kicinger 2004), the design method based on models of complex systems.

Once the values of all the parameters implemented in this group of components have been determined (default values are also used where possible), the actual design experiment is initiated. **Basic Statistical Analysis Component** and **Basic Dynamical Systems Analysis Component** allow the online monitoring of the design process by providing best-so-far fitness values and trajectories of the points (design concepts) in the design spaces. **Basic Statistical Analysis Component** also provides the mechanisms for the collection of relevant experimental data and saving them in files.

When the design experiment is finished, **Basic Statistical Analysis Component** can be used to calculate and display average best-so-far fitness values with corresponding 95% confidence intervals. At that point, the user can also generate a complete experimental report listing all the parameters and their values used in the design experiment as well as the experimental results. **Report Generation Component** and **Visualization Component** are employed during the process of the automatic generation of an experimental report. **Report Generation Component** gathers the names and values of the
parameters used in the design experiment and extracts relevant experimental results. It also collects important statistical data calculated by the Basic Statistical Analysis Component. Visualization Component can be used to produce a landscape visualization graph, if applicable, and charts representing progress of individual runs in the design experiments. When all the textual, numerical, and graphical data are available, Report Generation Component compiles them together into a single document that is subsequently displayed as an experimental report.

At this point, the user can choose to start a new design experiment, or to analyze the experimental data using advanced statistical and time series analysis tools, or simply exit the system. If a new design experiment is selected, Problem Definition Component is called again and the entire process described above is repeated. On the other hand, if advanced statistical analysis, or advanced time series analysis, is chosen then Advanced Statistical Analysis Component or Advanced Time Series Analysis Component is utilized, respectively.

3. System Components

This section describes in detail each group of the system’s components that were briefly introduced in the previous sections. The actual implementation of the system’s components is discussed in section 4.

3.1. DESIGN COMPONENTS

Design components implement Emergent Engineering Design (Kicinger 2004), the design method based on models of complex systems and inspired by the processes occurring in nature. Individual components included in this group correspond to the major phases of the design method which are shown schematically in Figure 3. The group of design components consists of four major components which are described below.

![Figure 3. Phases of the design method implemented in Emergent Designer](image)

3.1.1. Problem Definition Component

This component implements the preliminary phase of the design method in which a design problem is defined. The output of this component, i.e. a complete description of a design problem in terms of parameters and their
values, becomes the input to the Representation and Decomposition Component. This component provides necessary domain knowledge and specifies parameters of the considered design problem. It is used to perform the following tasks:

- Domain selection, e.g. steel skeleton structures in tall buildings.
- Problem selection, e.g. design of a wind bracing system, or design of the entire steel structural system.
- Specification of the problem parameters, e.g. the number of stories, or story height.

The external/user input to the component defines overall purpose (what to design), requirements, and constraints (feasibility criteria) the design should satisfy.

3.1.2. Representation and Decomposition Component
This component is used to conduct the first and second phases of the design method, i.e. Representation Space Definition and Representation Space Decomposition, in which representation of an engineering system and its decomposition, if any, are defined (see Figure 3). The input to this component consists of a complete definition of the design problem which is obtained from the Problem Definition Component. The output produced by the Representation and Decomposition Component defines the representation of the engineering system being designed. This component is used to conduct the following tasks:

- Selection of a representation for the design problem, e.g. parameterized representation, or generative representation based on one-dimensional or two-dimensional cellular automata.
- Selection of a decomposition of a given problem.
- Specification of parameters for a given type of representation (e.g., resolution for binary representations, or the neighborhood shape and the neighborhood radius for generative representations).

When all the decomposition and encoding parameters have been defined, the representation of the engineering system (artifact) is completely specified. The representation becomes the output of the Representation and Decomposition Component that is subsequently utilized by the Concept Generation and Optimization Component.

3.1.3. Concept Generation and Optimization Component
This component implements the third phase of the design method, namely Generation and Optimization of Design Concepts (see Figure 3). This component defines representations of the engineering design processes. The following tasks are handled using this component:
• Selection of the mechanisms of design concept generation, including various types of cellular automata (1D, totalistic 1D, 2D, totalistic 2D).
• Selection of the mechanisms of optimization of design concept, including various types of evolutionary algorithms, e.g. evolution strategies, genetic algorithms, etc.
• Specification of parameters for optimization algorithms, i.e. parent and offspring population sizes, types of genetic operators, etc.

Representation of the engineering system being designed is obtained from the Representation and Decomposition Component and forms the input to the Concept Generation and Optimization Component. The produced output consists of feasible design concepts with assigned fitness value(s).

The flow of information within the Concept Generation and Optimization Component is shown in Figure 4. It consists of three major subcomponents: Concept Generation Component, Topology/Shape Optimization Component, and Sizing Optimization Component. Depending on the type of representation of the engineering system provided as input and decisions made regarding the optimization mechanisms either only one subcomponent, or two, and even all three subcomponents can be utilized.

If only Concept Generation Component is used, then the design concepts are produced solely by the concept generation mechanisms, e.g. 1D or 2D cellular automata. In this case, no optimization mechanisms are employed during the design process. Generated design concepts are evaluated, given some evaluation criteria, and best designs are identified. Thus, in this case the focus of the design processes is shifted towards creativity/novelty.

On the other hand, if the engineering system is represented using parameterized encodings then no concept generation mechanism is necessary to produce design concepts from their representations. In this case, the design processes focus exclusively on optimality issues. Design optimization mechanisms can be applied at the topology/shape level (conceptual/embodiment design) using the Topology/Shape Optimization Component and/or member sizing level (detailed design) using Sizing Optimization Component.

It is possible to combine design concept generation and optimization mechanisms. In this case, creativity/novelty of generated design concepts and their optimality become equally important objectives. To achieve both objectives, concept generation mechanisms have to be defined using the Concept Generation Component and optimization mechanisms must be determined using the Topology/Shape Optimization Component and/or the Sizing Optimization Component.

Each produced design concept is tested for feasibility. When it satisfies all feasibility criteria defined by the Problem Definition Component then it is
passed to the *Evaluation and Simulation Component*, where it is evaluated and assigned fitness value(s). On the other hand, when a produced design concept is proved infeasible, then constraint-handling methods have to be employed. In the case when a repair algorithm is used, an attempt is made to repair the design concept and, if successful, the design concept is passed to *Evaluation and Simulation Component* and assigned fitness value(s). If the repair is unsuccessful, the design concept is determined infeasible and assigned worst possible fitness value(s).

*Figure 4. Information flow within the Concept Generation and Optimization Component*
The output of Concept Generation and Optimization Component consists of feasible design concepts with assigned fitness value(s) which are subsequently passed to the Basic Statistical Analysis Component, Basic Dynamical System Analysis Component, and Report Generation Component.

3.1.4. Evaluation and Simulation Component
This component implements the last phase of the design method, namely Fitness Evaluation. It defines design evaluation models and general mechanisms of managing and monitoring simulations of design processes. The following tasks are conducted using this component:

- Specification of the load conditions considered during the evaluation process, e.g. wind loads acting on a steel structure in a tall building.
- Selection of the evaluation model and mechanisms used to measure goodness of the generated design concepts, e.g. a structural analysis package.
- Specification of the overall simulation parameters, e.g. number of runs, lengths of individual runs, etc., and monitoring of the simulation progress.

The input to this component consists of a phenotypic representation of a design concept which is obtained from the Concept Generation and Optimization Component. The output produced by the Evaluation and Simulation Component consists of the same design concept, or design concepts, provided as input but this time with assigned fitness value, or fitness values in the case of multiobjective evaluation.

3.2. ANALYSIS COMPONENTS
Analysis components implement tools and methods for the analysis of the experimental results and design processes. The components included in this group are aimed to provide quantitative information about the conducted design processes as well as statistical estimates of the performance of the design method. They are also intended to provide deeper understanding of the dynamics of design processes and the structure of the design spaces from global/holistic perspective. A brief description of the four major components in this group is presented below.

3.2.1. Basic Statistical Analysis Component
This component implements basic statistical tools for the analysis of the results of design processes. The input to this component is obtained from Concept Generation and Optimization Component and consists of design concepts with their fitness values as well as their data regarding when they were generated during the simulation (their “birth dates”). The following tasks are performed using this component:
• Collection of the experimental data and calculation of the best-so-far fitness statistics for each design process.
• Calculation of various statistical estimates that quantitatively describe design processes, including average best-so-far fitness and confidence intervals around the mean.
• Comparison of statistical estimates (means and confidence intervals) calculated from the results obtained in different design experiments.

The first two tasks described above are performed online, i.e. during the actual design processes. When a new design concept has been generated and evaluated, its fitness value(s) and birth date are collected. This information is subsequently saved in the files storing the experimental results data. Next, the data are analyzed and best-so-far statistics are calculated. They are also saved in the files storing statistical analysis data. At the same time, best-so-far statistics are passed to the Visualization Component. When the simulation is over, the average best-so-far statistics for the entire experiment are calculated and saved in a file and they are also transferred to the Visualization Component. The output produced by the Basic Statistical Analysis Component consists of basic statistical analysis data which are subsequently passed to the Visualization Component and the Report Generation Component.

3.2.2. Basic Dynamical Systems Analysis Component
This component implements basic tools and methods for the analysis of the results of the design processes from the dynamical systems perspective. In this type of analysis, the design processes are considered as dynamical systems operating in the design spaces. The subjects of analyses are the properties of trajectories (coordinates of the generated points in the design space) of design processes and identification of attractors in the design spaces. The following tasks are conducted using this component:
• Collection of the trajectories data (coordinates of the generated points in the design space).
• Reconstruction of attractors in the design spaces from the experimental data.

These tasks are also performed online and show the actual dynamics of the design processes. The trajectory information is extracted from the collected data and passed to the Visualization Component. Further, the trajectory data are analyzed and methods and tools of reconstructing attractors are employed, e.g. delay coordinates. The results of these analyses are subsequently transferred to the Visualization Component. The output produced by the Basic Dynamical Systems Analysis Component consists of basic dynamical systems analysis data and is utilized by the Visualization Component.
3.2.3. **Advanced Statistical Analysis Component**

This component implements advanced statistical analysis methods, models, and tools for the analysis of the experimental results. The statistical analysis conducted by this component is performed offline when no design experiments are running. The input is obtained from the files storing the experimental results that were previously saved using the Basic Statistical Analysis Component. **Advanced Statistical Analysis Component** contains the tools for analyzing the sample distributions and making inferences about their means and medians. The following types of tasks can be conducted using this component:

- Reading the experimental data from file(s).
- Qualitative and quantitative analysis of the sample distributions, e.g. histograms, normal scores plots, skewness and kurtosis estimates, etc.
- Estimation of statistical quantities from the data, including means and medians, using various point estimates and interval estimates.
- Saving the results of the statistical analysis in a file.

The results of the advanced statistical analysis are transferred to the Visualization Component and subsequently displayed in the form of charts, graphs, and histograms and/or saved in the files.

3.2.4. **Advanced Time Series Analysis Component**

This component implements advanced tools and models from the linear and nonlinear time series analysis. The analysis, similar to the one performed by the **Advanced Statistical Analysis Component**, is conducted offline. Also, the input consists of the experimental results stored in the previously saved files. The following types of tasks can be conducted using this component:

- Reading the time series data from file(s).
- Qualitative and quantitative analysis of the time series data using various methods and tools, e.g. delay coordinates, power spectrum, autocorrelation, etc.
- Saving the results of the time series analysis in a file.

The output produced by the **Advanced Time Series Analysis Component** consists of the results of various analyses and is utilized by the Visualization Component and/or saved in the files.

3.3. **VISUALIZATION COMPONENTS**

Visualization components implement various visualization methods and report generation mechanisms. They contain tools for the visualization of the results of various analyses, e.g. statistical or time series, conducted by the system’s components, and simple fitness landscapes. Also, automated tools of generating experimental reports that include detailed descriptions of
experimental parameters and their values and the obtained results are implemented. There are two major components in this group:

3.3.1. Visualization Component
This component implements various methods of data visualization. It is aimed to provide a qualitative analysis of the experimental results and the necessary functionality to save produced graphs and charts in files and experimental reports. The input to this component consists of the experimental data and is obtained from various components of the system. The following types of tasks can be conducted using this component:

- Display of generated design concepts.
- Interactive display of simple three-dimensional fitness landscapes.
- Display of statistical, dynamical, and time series analyses conducted using various components of the system.

The experimental results obtained as input to this component are collected and information relevant to display and visualization purposes is extracted from the data. Next, depending on data source, appropriate graphs and charts are produced including line charts, scatter plots, histograms, and renderings. The produced graphs are next displayed by Emergent Designer’s graphical user interface (GUI). Each generated graph may also be saved in a file. This last option is implicitly used by the Report Generation Component which utilizes various graphs produced by the Visualization Component during the process of automatic generation of the experimental reports. The graphs included in the reports are first saved in files and subsequently read by the Report Generation Component.

3.3.2. Report Generation Component
This component implements mechanisms for the automatic generation of experimental reports. It is intended to provide complete information about the experimental parameters and their values as well as the obtained results. The input to this component is obtained from various components. The following types of tasks can be conducted using this component:

- Collection of the experimental parameters and their values used in the reported experiment.
- Collection of the numerical results of various runs in the reported experiment.
- Collection of statistical analysis data and various graphs illustrating progress of individual runs and average performance during the entire design experiment.
- Automatic generation of a full report containing all above mentioned elements.
The output produced by the Report Generation Component consists of a complete experimental report which is displayed in the system’s GUI and/or saved in the file.

4. Implementation

*Emergent Designer* has been implemented with fully functional graphical user interface using Java. The decision to use this particular programming language was made due to the fact that several of the system’s components were built upon existing packages written in Java. Moreover, *Emergent Designer* integrates several commercially available systems (e.g., Mathematica© (Wolfram 2003) and OpenOffice.org©) and communicates with them using available Java APIs.

Another important aspect that influenced the choice of the programming language was the fact that Java is portable and network-oriented. Portability offers the flexibility of running the system on various platforms. Built-in networking capabilities open the possibility of using distributed architectures. Both of these issues are particularly important in structural design where the process of evaluation of generated design concepts is usually computationally expensive and conducted using specialized structural analysis software.

4.1. DESIGN COMPONENTS

Components implementing the design method constitute the core of *Emergent Designer*. The functionality described in sections 3.1.1 – 3.1.4 was either directly implemented or borrowed from several existing packages and commercial systems that were integrated with *Emergent Designer*.

Two domains have been implemented in the Problem Definition Component: the domain of steel structural systems in tall buildings and the domain of real-valued functions (added for testing purposes and analysis of the behavior of various components of the system). The domain of steel structural systems includes two major classes of design problems: design of a wind bracing system in a tall building and design of the entire steel structural system in a tall building.

*Representation and Decomposition Component* allows four types of representations: binary, real-valued, integer-valued and cellular automata. The first two types are used mostly for real-valued problems while the latter two are applied to encode the designs concepts of steel structural systems in tall buildings. Real-valued and binary representation implementations were inherited from several existing evolutionary computation packages, including ec1, ec2, and ec3 (De Jong to appear). On the other hand, integer-valued and cellular automata representations were directly implemented in the system.
Concept Generation and Optimization Component has been built upon four major existing packages and commercially available systems. Design concept generation utilizing various types of cellular automata is conducted by Mathematica® kernel which was integrated with Emergent Designer via JLink™. All major types of cellular automata (CA) are supported, including 1D CA, totalistic 1D CA, 2D CA, and totalistic 2D CA.

Topology/shape optimization using an evolutionary algorithm (EA) is performed by ec3 package (a Java-based EC toolkit (De Jong to appear)). Here, all canonical evolutionary can be utilized, including genetic algorithms, evolutionary programming, and evolution strategies. The system also offers an option of employing a unified EA (De Jong to appear) in which all major elements the EA, i.e. generational model, parent selection, offspring selection, population sizes, operators, etc., can be tuned to the particular problem being solved.

Sizing optimization, if applied, is conducted using an optimization algorithm based on traditional mathematical programming method and implemented in SODA. It is a commercially available structural analysis, design and optimization system developed by Waterloo Systems in Waterloo, Ontario, Canada which was integrated with Emergent Designer to perform evaluation of designs and their sizing optimization.

Evaluation and Simulation Component implements evaluation models used to determine fitness of generated solutions. Current status of the system allows only for a single objective evaluation of individual design concepts using one of the following evaluation criteria: the total weight (an estimate of the cost) or the maximal horizontal displacement (an estimate of the stiffness) of the steel structural system. The determination of a least-weight structure is performed by SODA and is conducted in conformance with the strength (stability) and stiffness (displacement) provisions of several commonly used steel codes, including AISC-ASD-89, AISC-LRFD-86, AISC-LRFD-93, CSA-S16.1-M89, or CSA-S16.1-94. Loading model required for evaluation of generated design concepts includes dead, live, and wind loads determined in conformance with the corresponding design codes. Wind forces are calculated for given design cases using a modified commercial system Wind Load© V2.2.S developed by Novel CyberSpace Tools.

Graphical user interface of Emergent Designer displaying Representation and Decomposition Component is presented in Figure 5.
4.2. ANALYSIS COMPONENTS

Methods and models for basic statistical and dynamical systems analysis have been implemented directly in Java. The simple analysis is conducted online, i.e. during the actual design processes. Basic statistical analysis involves best-so-far fitness statistics calculated for individual runs and average best-so-far fitness statistics and 95% confidence intervals computed for the entire design experiment. This analysis is automatically saved in files.

Implemented methods of simple dynamical systems analysis include trajectory analysis which shows the dynamics of the processes in the design spaces as well as delay coordinates analysis. Delay coordinates are computed from the best-so-far fitness values with an arbitrarily assumed time lag.

Advanced statistical and time series analysis is performed offline, i.e. when the design processes are completed. Advanced statistical analysis includes estimation of sample distributions using histograms, normal scores plots, symmetry plots, and estimators of sample kurtosis and sample skewness. It also implements various estimators of means and medians of
the sample distributions and their corresponding confidence intervals. Implemented mean point estimators include: the sample mean and the trimmed mean. Confidence intervals around means can be computed using the following three methods: normal approximation, Student’s t test, and Johnson’s modified t test. Point estimates of medians may be calculated using the sample median and the trimmed mean while the confidence intervals around median are determined by the following two methods: sign test (Thompson-Savur formula) and normal approximation (using the conservative approach). Advanced statistical analysis tool and methods have been in part implemented directly and partially borrowed from JMSL® Numerical Library that was integrated with Emergent Designer.

Advanced time series analysis offers the following methods of analysis of the experimental data: visual analysis of the time series data, delay coordinates plots with adjustable parameters determining the embedding dimension and the time lag, power spectrum analysis, autocorrelation analysis with a flexible specification of autocorrelation lag and standard error bars according to either Barlett’s or Moran’s formula, and two types of recurrence plots: regular and thresholded with a flexible specification of the embedding dimension, time lag, and norm to calculate the distances between the points of a time series.

Also in this case, several tools and methods of advanced time series analysis were directly implemented in the system while several have been borrowed from JMSL® Numerical Library.

4.3. VISUALIZATION COMPONENTS

Experimental data are visualized in Emergent Designer in three major ways. First, line plots and scatter plots (or more generally signal plots) are used to visualize experimental data transferred from the Basic Statistical Analysis Component and Basic Dynamical Systems Analysis Component. These types of graphs include best-so-far fitness plots, average best-so-far fitness plots with error bars, trajectory plots, and delay coordinates plots. The plots are produced by a Java-based signal plotter called PtPlot developed at UC Berkeley. They are embedded in the Emergent Designer’s graphical user interface and can be subsequently saved as bitmap files (in eps and png formats). Histograms are another type of graphs generated during the analysis conducted by the Advanced Statistical Analysis Component. These types of graphs are created using JMSL® Graphical Library integrated with Emergent Designer. They are also embedded in the system’s GUI and provide functionality to save the produced graphs as bitmap files. Finally, interactive renderings of simple real-valued fitness landscapes can be produced. This type of visualization is produced using Mathematica’s
advanced graphical capabilities. Generated renderings are displayed in the system’s GUI using JLink.

Automatic report generation capabilities have been achieved by the integration of Emergent Designer with OpenOffice.org using its Java API. Report Generation Component collects and organizes the textual, numerical and graphical information that should be included in the experimental report and then creates an OpenOffice.org document which is subsequently displayed on the screen. The report can be later saved as a file in any of the format supported by the OpenOffice.org and thus provide complete summary of the experimental parameters and the obtained results.

Figure 6. Visualization and report generation capabilities of Emergent Designer

5. Conclusions

Emergent Designer is a complex design support tool which belongs to a new generation of design tools being developed at George Mason University. They result from several years of intense research on evolutionary designing, on complex adaptive systems, and on cellular automata in the context of New Kind of Science as proposed by Wolfram (2002). The developed system is intended for research purposes, but in the future it will be adapted for the practical engineering design applications.
The initial experience with *Emergent Designer* has clearly demonstrated that the concept of the system was feasible. The system provides fascinating research opportunities related to evolutionary design and cellular automata. It is too early to provide a balanced evaluation of the system’s advantages and disadvantages, but the authors believe that its use will soon lead to various discoveries related to design and engineering creativity, particularly related to emergence.

**References**


Kicinger, R: 2004, Emergent Engineering Design: Design creativity and optimality inspired by nature, Ph.D. Dissertation, School of Information Technology and Engineering, George Mason University, Fairfax, VA, USA.


Wolfram, S: 2002, A new kind of science. Wolfram Media, Champaign, IL.

GEOMETRY AS A SUBSTITUTE FOR STRUCTURAL ANALYSIS IN GENERATIVE DESIGN TOOLS

MARTIN HEMBERG, UNA-MAY O'REILLY
Imperial College, UK and MIT, USA

Abstract. Structural analysis using finite element methods is, in general, a non-trivial method that is hard to integrate within a versatile design tool and it is often costly in terms of computation. In this paper we offer anecdotal evidence that, with a designer’s understanding of the relation between geometry and structure, purely geometric concepts can complement and frequently substitute for structural analyses in creative design tools. In two systems, Genr8 and Surface Component System, geometry has not just often stood in for FEA, it has integrated structural considerations within a multi-objective creative interactive design process rather than tack them on as a later, independent design step.

1. Introduction

In Emergent Design (ED) generative algorithms are used to create complex artifacts, often inspired by organic growth and form. Important properties of an architectural design are frequently structure and material. Engineers have developed sophisticated methods for analyzing the structure and stability of designs, many based on finite element analysis (FEA). The authors’ work on ED has focused on form finding and, to date, we have ignored structural analysis to a large extent. Our long-term goal is to be able to integrate the two and allow them to inform each other during the process. In this paper we argue that the purely geometric analysis used in form finding can also be useful in terms of structural analysis.

Performing a FEA is a non-trivial task and the designer must specify a range of parameters in order to analyze the specific scenario. Moreover, it is often computationally costly to conduct an FEA and interpreting the result requires considerable effort in comprehension by the designer. If one uses a creative design system with an EA component, the computational cost can become significant due to the large number of evaluations required.

In this paper we discuss how structural analysis can be incorporated in creative design tools (Bentley and O'Reilly 2001) by means of a purely geometric analysis. We define a Creative Design Tool (CDT) as a tool that includes a generative growth algorithm and/or an evolutionary search algorithm (EA). We will use the CDTs Genr8 (Hemberg 2001) and Surface Component System (SCS) (Jonas 2004) to illustrate the concept. A complete overview of CDTs is beyond the scope of this paper and a brief survey of CDTs can be found in Hemberg and O'Reilly (2004b).
2. Creative Design Tools

Today’s desktop computers are powerful enough to be used for more than traditional CAD tasks, but in order to achieve that goal, novel algorithms and software are required. Regardless of such algorithms and software, the human designer should have full creative control of the design process. CDTs should be creative, interactive, cooperative and easy to integrate in a design process. Geometric analysis is useful to achieve this and should be exploited in order to give creativity and exploration priority over structural analysis and optimization. The authors have been involved in the development of a number of tools where the main focus has been on form generation. Focusing on form generation allows us to better concentrate on developing creative algorithms.

We feel it would be incorrect to incorporate structural fitness into a CDT as a final step that, at one design point, provides detailed analysis and local optimization of the almost finished design. When a designer employs a CDT, a process is undertaken and multiple criteria are juggled and considered. Structural performance must simply be one of these so that a designer can also take aesthetic considerations and functional aspects into account.

3. Geometric Constraints

In a CDT structural analysis and creative form finding might appear to be orthogonal. Furthermore, any introduction of structural analysis must not impede the technical aspects of the generative algorithm. Our basic concept is to use purely geometric criteria as a substitute for the structural analysis. When the FEA is complicated and costly, it pays off have simpler purely geometric evaluation criteria as a complement.

Thus, we propose an evaluation process consisting of two interleaved stages. In short, geometric criteria are evaluated and used to drive evolutionary adaptation several times before an FEA is invoked. The first stage contains only geometric criteria. These are typically fast and easy to evaluate and they can provide a rough guidance towards structurally sound designs. This stage is used initially and much more frequently. The second stage executes a more detailed structural analysis (possibly based on FEA). It runs less frequently than the geometric stage, but is selectively used on designs that have been coarsely determined to be structurally sound based on geometric criteria and which have been arrived at by considering multiple other design criteria.

EAs require a fitness evaluation procedure and a common way of assigning fitness in creative design tools is to have the user examine and rank or select each member of a very small population. This scheme, called Interactive Evolutionary Computation (IEC) restricts the evolutionary search due to human fatigue (Sato and Hagiwara 2001). We prefer the use of computational, parameterized fitness criteria that express geometric features and providing a partially automated control of the evolutionary search (Hemberg and O’Reilly 2004a) thereby covering a greater portion of the search space. The designer is allowed to set target values for the properties as well as weights that indicate how important each property is (examples can be found in Table 1). We shall put the onus on the users to make a connection between these criteria and the structural analysis despite it not always being a straightforward or trivial task.
To explore the potential of a CDT, when users specify the fitness criteria they must ensure that the criteria vary non-linearly and they are promoting non-linear search. Our experience indicates that the most interesting results are obtained when the criteria are chosen in such a way that the parameters cannot be independently optimized to reach the global optimum. The significant consequence of parametric non-linearity is that there are many different surfaces of equal fitness (i.e., a Pareto-front), which implies diversity. It also presents the designer with provocative and interesting trade-offs to make between the different criteria since deprecating the importance of one or more fitness criteria is often required to improve overall fitness.

In comparison with FEA, geometric analysis is independent of a specific context or scenario and material properties and can be thought of as more general. The benefit of involving structural criteria through the definition of geometric surface features is that the user can express her understanding of their performance in relation to a specific need or scenario. (Albeit, to take structural advantage of those criteria, the user must have understand how structural geometry works. For example, knowledge that folded surfaces are generally stiffer than flat surfaces.) Note that geometric criteria do not have to be specified as fitness criteria, one could consider other ways of imposing constraints. For example, one could use boundary boxes that are either impenetrable or penetrable at a fitness penalty. Geometry is also advantageous because it affects multiple aspects of the design, (e.g., environmental and visual) not only structural ones. Use of a geometric abstraction, in short, makes the CDT more versatile and easier to apply to a wider range of design tasks.

4. Experience From Using Design Tools at the Architectural Association

Work that has been conducted in the Emergent Design + Technologies group (2004) at the Architectural Association in London shows that it is possible to create interesting designs using the strategy outlined in this paper. Achim Menges of the AA has used Genr8 to conduct experiments.

In one of his experiments, Menges explored the interaction between a form finding process based on evolutionary dynamics and a computer-aided manufacturing process. The aim of the project was to design a pneumatic strawberry bar (see Figure 1) for the AA’s annual project review. Menges formulated a number of form and material criteria that had to be realized through Genr8’s fitness function. A complex feedback loop was set up where the three subpopulations were evolved simultaneously. Menges exploited the ability to model environmental features in Genr8 by making earlier evolved surfaces into environmental bounding boxes to help realize the constraints. Part of the structural analysis was conducted in a dedicated membrane engineering software, EASY. The results of this analysis were interpreted by Menges and he used them to modify the parameters for his Genr8 runs. Menges’ experiments are described in greater detail in O’Reilly et al. (2004). When Genr8 was developed, this strategy was not anticipated, Menges came up with it independently. His ability to take such initiative shows the value of creating a versatile tool that allows the user to integrate them into a larger design process.
Another tool, a Surface Component System (Jonas 2004) developed by Katrin Jonas with the assistance of Martin Hemberg implements the strategy outlined in this paper in full (Jonas and Hemberg 2004). The tool uses a genetic algorithm to combine tiles from a predefined set to form surfaces. The surroundings of a grid space determine what subset of tiles can be used for that site. The set of tiles is shown in Figure 2 and two examples of such surfaces are shown in Figure 3. The surfaces in Figure 3 show how geometric fitness criteria can be used to alter structural properties. Both surfaces are evolved to have the same number of support points. However, the lower one is clearly more useful since the support points are not clustered in one region. This was achieved through a purely geometric fitness criterion that simply specified the desired distance between the support points.

Part of the tool is implemented as a MEL script for Alias|Wavefront’s 3D modeler Maya. The geometric fitness criteria are evaluated inside Maya and they are presented in Table 1. The user must experiment with the target values and the weights of the criteria in order to find suitable values. By inspecting the outcome, the user can learn how to adjust them in order to achieve the desired result for the design task at hand.

Figure 2. The set of 17 tiles that are used in the Surface Component System. There is a set of rules that specify which tiles are allowed to be adjacent in order to create a continuous surface. It is possible to create a situation where the set of allowed tiles becomes empty and in that case a hole is left in the surface.

| Table 1. The geometric fitness criteria for the Surface Component System. The criteria are fast and easy to compute and they are intuitive for the user. The user can |
set weights and target values for each criterion. The last criterion is illustrated in Figure 1.

<table>
<thead>
<tr>
<th><strong>Fitness criteria</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>The difference between the highest and the lowest points in the surface.</td>
</tr>
<tr>
<td>Number of holes</td>
<td>Sometimes it is impossible to select a tile that fits in a location. In those cases, that position becomes empty.</td>
</tr>
<tr>
<td>Number of support points</td>
<td>How many different points of the surface are at the minimum z value.</td>
</tr>
<tr>
<td>Support point distance</td>
<td>In order to avoid having all the support points clustered in one region, we impose a target distance between all support points.</td>
</tr>
</tbody>
</table>

*Figure 3.* The figures show surfaces created with the Surface Component System. In the top figure, there is no selection pressure to have the support points (marked by red dots) separated and the tool finds a solution where all points are clustered in the same region. In the lower figure, a fitness criterion that regulates the distance between the support points has been introduced. It is clear that the new fitness criterion helps create surfaces that are structurally more sound.

After a number of generations, the surfaces are exported to the ANSYS software for further FEA. The output from the ANSYS analysis is a ranking of the surfaces (the fitness of the surfaces can only be stated in relative terms) that can be used to inform the selection process of the evolutionary algorithm. An example of a stress analysis is shown in Figure 4.
5. Summary

Our solution to structural analysis is to let the user communicate structural concerns via choosing the parameters of the fitness functions and setting up the environment to take structure into account. While this is not always a trivial and straightforward task for the designer, it pays off beyond avoiding sparing use of computationally expensive, hard to interpret methods such as FEA. It pays off because, when structural analysis is integrated with other non-linear design considerations, rather than after they have been resolved, it has the opportunity to inform and greatly influence the design process. This implies that more creative and interesting design potentials are available. In the context of work that has been conducted in the Emergent Design + Technologies group (2004) at the Architectural Association in London, we have demonstrated the feasibility and payoff of our solution in two different tools, Genr8 and Surface Component System with two different projects and design goals.

Acknowledgements

The authors would like to thank Achim Menges and Katrin Jonas for valuable discussions and comments when preparing this manuscript. We are also grateful for them providing us with the images used in the paper.

References

Hemberg, M and O'Reilly, U-M: 2004b, Using developmental growth to create architectural
design, submitted to the Workshop on development in natural and artificial systems,
Alife9.
Jonas, K: 2004, Component surface system – system tool (the application of an evolutionary
Jonas, K and Hemberg, M: 2004, Integrating structural analysis and evolutionary computation
in the surface component system, Manuscript in preparation.
O’Reilly, U-M Hemberg, M and Menges, A: 2004, Evolutionary computation and artificial
life in architecture; exploring the potential of generative and genetic algorithms as
Sato, T and Hagiwara, M: 2001, Idset: Interactive design system using evolutionary
ARGENIA, A MOTHER TONGUE IN INFINITE VARIATIONS

Identity rises from a soundless site, because birds have no tears.

CELESTINO SODDU, ENRICA COLABELLA
Generative Design Lab, Politecnico di Milano University, Italy
celestino.soddu@polimi.it
everica.colabella@polimi.it

Abstract. Argenia is a generative AI software built in 1988 and upgraded following the design activity. It works producing endless variations of Artificial DNA as generated species (fig. 1). In this paper the authors present the complex structure of generative systems and some Visionary Variations as a poetic product. Also some teaching results using the generative methodology will be showed.

Keywords: AI, Generative Systems, Variations, Visionary, Poiesis, Creativity, Abduction, Architecture, Naturality.

Figure 1. Generated Variations of Castles.
1. IDEA AS ARTIFICIAL DNA, a generative system based on a set of codes of transformation

The design Idea is represented with creative algorithms that define multiple paths from an existing world towards possible worlds. Idea is like a natural DNA, a transformation code that identifies a lot of different relationships starting from the designer’s idiom. These algorithms can be used in a creative complex process for every project and it is possible to upgrade them as an increasing experience. So we define a set of not linear procedures (words, matrixes, schemes, paradigms) able to perform open generative systems. This creative methodology brings to formalize an artificial Dna, which is based on the observation of the environment that surrounds us and on our cultural references.

Tools for Idea
This first step of the generative process consists in two different moments:

1. Abduction

As happens in every creative act, such observation activates a fixing up of our impressions, as a performing of our subjective interpretation. This procedure was defined by Pierce as an abduction process, able to define an “how to reason”. The interpretation is an experience. The aim of the formal logic is to delineate a table of categories, able to be a faneron, a manifestation of experience. We can read and record what fascinated us as a code of transformation. Hume described this mental process as an idea that reminds an impression about an absent object. In this way what is observed and appreciated becomes a strong deep indication for identifying a process of transformation that will operate on the existing environment to transform it into a possible scenario. So we call abduction this type of performing rules because the interpretation of the events is managed with the aim to represent a rule as an algorithm.

The construction of an artificial Dna is also a basic process that can grow toward complexity using algorithms as a useful answer to the contemporary needs of Quality and Beauty.

2. Attributes

We use creatively our mother tongue defining some words as attributes, minimum three or more, as a possible significance of our iter. This is a very important tool that connects words with pictures in a logical procedure, as in ancient time, ut pictura poiesis. But this is also peculiar to the classic music. We have a lot of different attributes as indications for playing Mozart, Bach Rossini etc. The author can gain an open number of variations with these multiple topic suggestions.
ARGENIA, A MOTHER TONGUE IN INFINITE VARIATIONS

2. PERFORMING SYSTEMS

Argenia, our generative system, essentially consists of two different components: a system of codes of transformation and an organizational paradigm able to control the evolutionary dynamics. Both elements are absolutely essential for realizing an idea/code as a generative motor. These parts produce together a generative non-linear system.

2.1. THE ARTIFICIAL DNA

The code system can be compared to the natural structure of DNA. The whole structure of rules is a matrix device that defines how to evolve a system towards complexity. In other words, it is possible to consider it as a group of concepts that trace and identify a specific design behavior. In the case of Argenia, since in the first years of study it was deepened and developed as an act of author’s interpretation of the world of the Renaissance Masters. These rules mirror author’s concept of the Beauty of Nature. They represent the subjective fascination for some particular natural structures, whose harmony is translated into rule systems. These rules are not objective; neither do they originate from a philological approach. As a whole, they represent author’s concept of harmony, which was born from a personal and cultural experience, from a reading of the universe of natural and artificial events where a new vision of “reality” can rise. This approach is close to the Renaissance spirit and to its challenge: the interpretation of natural harmony as an operational code for realizing an artificial world, “la città ideale”. They perform a poetic structure.

This is a set of transformation codes of evolutionary logics that comes forth from tradition and that mirrors author’s identity. Again, here one comes across the inseparable synergy between the artistic and scientific approach. “Ars sine scientia nihil est” In fact, by experimenting with the evolutionary logics of complex systems, we are able to look for fragments of our code of harmony fathoming the Possible, seeking for our idea of beauty. The system’s non-linearity, and its realization of unpredicted exceptions, represents an unexpected synergy among non-connected parts of our aesthetical research.

In searching for rules as codes of harmony, we have experimented with geometric, perspective and mathematical codes, which were visualized and made operational by computers. In particular, we have made transformations by using imaginary numbers, what has enabled the author to identify the role
of events within a complex system. In comparison to parallel events, this system could be transformed and evolved along one “preferential path”, which, through possible variations of the subsystems and of the details that represent it, implies a logic of paradigmatic control over the whole and its identity. In Argenia it was also used multidimensional cellular automata for writing rules that define the topological structure of generated architectural scenarios.

TABLE 1. The generative process (cycle 1-2-3-4-5) uses algorithms managing the transformation and evolution inside a non-linear system, but not only the evolution of the system itself. Each generative project can generate, using a lot of parallel artificial lives, an endless sequence of possible parallel results for fitting the architect’s imprinting with the client’s needs.
In order to build individual events, this artificial DNA needed an evolutionary system, an artificial life, which allowed it to develop and achieve the levels of complexity proper of our time. The goal was the figuration of complex events, such as cities, buildings, industrial objects. It was not enough to produce beautiful and fascinating forms that allude to the natural complexity of possible environments, as for instance fractal or numerical wholes, represented by Bezier curves. The challenge was to produce artificial individuals that were "recognizable" within the complexity of the existing events: historical cities such as the Italian medieval cities, New York whose identity is also recognizable in the most marginal areas, Chicago with its ability to represent the history of the architecture of the twentieth century, Hong Kong with its unrepeatable mixture of west/east styles, and so on. This experimentation begun in 1986, starting from the Italian medieval cities.

2.1. THE EVOLUTIONARY PARADIGM
In order to reach the figuration of complex events, it was necessary to work out paradigms that used and guided the evolutionary structure of the system. This gave rise to results that, because of their difference, were typical examples of Italian medieval cities. In order to create these organizational paradigms, it was necessary to systematize the structure of the architectural space, as Renaissance architects have done. Therefore it was built a system in which relationships among architectural events are nested inside each other. This system formed a basis for the evolutionary structure and its exceptions. (Exceptions are important because they could overturn deeply the same structure during the evolutionary path.)

The system is made up of event-spaces with, all around, 26 synapses, which manage the interface with possible topologically near events. The whole system is therefore based on the number 27. Every direct relationship between two spaces can maximally activate 9 complex interfaces. Every spatial event therefore has one to nine parallel possibilities for structuring a relationship with the topologically near event. Every interface, to operate its own evolution-transformation, is characterized by a series of parallel systems which are usable together and which can be reciprocally contaminated. Each of these systems responds to specific characters that the event will have to explain and show. Each one has a different geometrical-topological structure representing its peculiarity. For instance, one of the evolutionary contaminations that has been used in some architectural generative projects works with three parallel geometric systems. The first one is based on the number nine (9x9x9 references linked with specific relationships, such as the golden section), the second on the progressive division of axis and diagonals, and the third one replaces the Cartesian coordinates by the polar ones building an order of preferential relationships.
with centre. The contaminations and the interferences between these three parallel systems simultaneously generate complexity and harmony. The system, in addition, provides some foldings that can alter the structure of topology making to coincide in a single event the interface produced by synapses, which are, before folding the system, distant one from the other. In this entire transforming path, time enters as a conclusive factor of the possible bifurcations of the evolutionary structure as it is the only unpredictable element. Every possible start-up of generative paths is conditioned from the temporal moment of its beginning, that is obviously always different. And the folding of the system, with the respective acceleration or induced decelerations, manages the unpredictability of the results.

This approach makes it impossible to repeat the same generation. This uniqueness is due to the temporality of the generative process that was started up, and it can only be overcome by producing clones through manipulating and resetting the clock, which is, of course, antithetical to the aim of the generative approach. Once the architectural spatial net is systematized, it was built, referring to every single design occurrence, the logical plot to check the evolutionary dynamics of the system in order to fit the client’s needs. In other words, if the generative project has to realize a multifunctional skyscraper in Los Angeles or, for instance, a chair, it is necessary for the codes of harmony to work in such a way, as to reach this goal. This part of the generative project is obviously built ad hoc for every single project. Therefore, schematically, we do not have a homologating technology.

The generative projects are not tools for producing any possible result. They can't be used by all creative people. The generative projects need a subjective approach that can answer the specific needs of the client, in line with the humanistic tradition. It is possible, instead, to define a modus operandi and to make it executable through rules, as we have experimented in didactics.

In our teaching activity, especially as supervisors of master thesis, we asked to our students to design their own artificial DNA based on their cultural references. This is an exercise for representing their idiom as a starting point in their creative process.

Three examples of master thesis.

1. The drawings of Piranesi as a representation of a poetic infinite space. The passage from 2D drawing into 3D models needs a subjective interpretation. This is a fixing up of a subjective unique and un-repeatable idea of space. Master thesis of Enrico Mazzei. (fig.2).

2. By using abduction the reconstruction of some transforming rules of Gaudí. The result was an artificial DNA able to generate some sequences of possible casa Batlo and casa Milà and evolving also hybrids with modern movement authors for delineating the borders of identity. Each scenario is
different but each one is the representation of the performing Idea of Gaudi
architecture. Master thesis of Matteo Codignola. (fig. 3).

Figure 2. Abduction from Piranesi. 3D models generated from different point of
view.

Figure 3. Abduction from Gaudi. The generated model of Casa Milà and two
variations of generated Casa Batlo.
3. The reconstruction of Trimellone Island, destroyed by a blast, designed with historical transforming rules. The historical and contemporary references are clearly represented as transforming rules and this generative process used these rules as codes in the architectural project of reconstruction. The master thesis of Germano Scalinzi (fig.4).

![Figure 4. Abduction of transforming rules from the medieval castle in Verona used in Trimellone Island reconstruction.](image)

3. ARGENIA, THE GENERATIVE SYSTEM

Argenia was designed as concept software that can generate architectural 3D models, and that controls dimensions, materials, technology and structure of function. It is therefore like a personal virtual architectural office. Each 3D model is generated from “harmonic codes”, which represent a strong identity and follow author’s architectural concept.

Argenia is a generative project that is always in progress. [In the fifteen years that it was used in our design activity, each occasion has been used to implement it in architectural projects. Now it is a real virtual office that can realize an impressive sequence of architectural 3D projects in real time. These projects define a dynamical vision, and can fit exactly the client’s needs. Constraints are not considered as problems, but as occasions for reaching a stronger identity in the artificial life of the project. Something similar occurs in nature. The more the needs of the clients are complex, the more the generative project can increase its quality and identity. Architectures must not only adapt to contemporary clients and users, but also to future users. This objective could be realized by giving the client the...
opportunity to choose among endless possible generated scenarios. At the same time a further objective exists: meeting the expectations of the city and its inhabitants. Can architecture, with the strong imprinting of its designer, increase the identity of a city? To verify this, we presented the inhabitants of cities with a strong identity, such as Hong Kong, a series of visionary images of their city, and it was asked them a very simple question: which of these visionary variations of Hong Kong is more like Hong Kong than before? It seemed that it is really simple for human beings to recognize similarity. Artefacts imitate nature.

The following images present a series of visionary scenarios of transformations in Hong Kong (fig.2), Washington DC (fig.3), Los Angeles (fig.4), New York (fig.5), Milano (fig.6 and 7), Shanghai (fig 8 and 9), Beijing (fig.10), Tianjin (fig 11), Nagoya (fig 12) and Chicago (fig.8). In these generative projects the architecture was designed and generated with Argenia, and tries to increase the identity of towns. These images were used in exhibitions to verify the impact on its inhabitants.

Figure 2. Generated Variations of buildings for Hong Kong waterfront.

Figure 3. Generated variations of the IDB new Cultural Centre, Washington DC
Figure 4. 1 real + 2 generated Variations in Beverly Hills

Figure 5. Completely generated town system representing the NewYork identity code.
Figure 6. 1 real + 2 generated Variation of a New Gallery in Milan

Figure 7. 1 real + 3 generated Variation of Milan with the New Museum of Futurism
Figure 8. Real environment and first scenario of generated two towers in Shanghai

Figure 9. Real environment and second scenario of generated two towers in Shanghai

Figure 10. 1 real + 2 generated Variations of a multi-purpose building in Beijing
A consideration: Argenia identifies a universal language that uses infinite variations as in Music and that every people can recognize because it starts from a simple singular mother tongue.

*Run, little boy, found the river in your mind*
References

Websites:
www.generativedesign.com
www.generativeart.com
www.celestinosoddu.com
master thesis:
www.artegens.com/lesi/005/lesi.htm
www.artegens.com/lesi/099/start.htm
www.generativedesign.com/lesi/111/indice.htm

Colabella, E: 1999, Imaging Humanity, Proceedings, USA, Bordighera Press, The imprinting of a man, word as medium for icona (pag.129)  The experiences of the students of Enrica Colabella, a. a. 1997/98 Corso di Rilevamento e Rappresentazione , Faculty of Architecture, Politecnico di Milano. Codes as expression of an hypothesis that comes from the Poetic Test of Giacomo Leopardi. The goal is the building of possible “real” scenerios of Architecture, that represent the formulated idea.

Pierce Charles Sanders “Categorie”, a cura di R. Fabbrichesi Leo, Ed. Laterza,1992


Colabella, E, Soddu, C,"Il progetto ambientale di morfogenesi", (the environmental design of morphogenesis), Progetto Leonardo Publisher 1992

Soddu, C, Colabella, E, "Spatial and image dynamical models for controlling and planning the development of differences in an increasingly international and interdependent world", in the proceedings of the conference "Planning in a time of Change", Stockholm 1992

Soddu, C, Colabella, E, "The project of morphogenesis to design the DNA or architecture", in the proceedings of "CIB'92, World Building Congress", Montreal 1992.

Colabella, E, Soddu, C, "The morphogenetic design as an artificial intelligence system to support the management of design procedures through the total quality of the built-environment", in the proceedings of "The management of Information Technology for Construction. First International Conference" Singapore, 17/20 August 1993.

Soddu, C, "The design of morphogenesis to design the genetic code of the artificial environment", in 33rd European Congress of Regional Science Association, Moscow, August 1993.


Colabella, E, Soddu, C, "Experimental CBL original software to teach project approach and designing procedures in architecture and industrial design", in the proceedings of the International Conference of Computer Based Learning in Science", Wien 18/21 September 1993.
Soddu, C, "Il progetto di morfogenesi / Design of morphogenesis", in the magazine Demetra, n.5 December 1993. (In Italian and English)


Colabella, E, Soddu, C, "Artificial Intelligence in the teaching of architectural design", in the proceedings of the conference Hypermedia in Sheffield, Sheffield 1995


Soddu, C, "Argenia, the natural approach to Architecture and Industrial Design" in the proceedings of Mecatronic workshop at the University of Lancaster, 27/28 May 1998.

Soddu, C, "GENERATIVE ART", proceedings of GA'98, Editrice Dedalo Publisher, Rome 1999


Soddu, C, "From Forming to Transforming", proceedings of Generative Art Conference, Milan, AleaDesign Publisher, December 2000


Duncan Graham-Rowe, Design Genes, Blueprint Magazine, August 2001

Soddu, C "Generative Natural Flux", proceedings of Generative Art Conference, Milan, AleaDesign Publisher, December 2001


Genometry: a genetically inspired parametric form generation method.

ANANDASIVAM KRISHNAPILLAI
National University of Singapore, Singapore

Abstract. This paper proposes a generative parametric design method based on creating variations around a generic design form. This method is suitable for a class of design problems where desirable solutions lie within close geometric proximity to known design solutions. In contrast to the conventional design strategy of specify, generate and evaluate, it is implemented in the reverse direction; by first setting up the generic parametric shape and by generating variations which may be evaluated by analytical or manual methods. A method of implementing such a design process is developed, with examples to demonstrate its efficacy in generating a wide variety of form designs.

1. Introduction

Form Design constitutes the basis of a very wide range of design activity, ranging from sculpture to mechanical engineering design with its practice based on a range of creative and analytical techniques. Form Design - though it may not be called as such – plays a very significant role in engineering as it determines the structural, mechanical and electrical behavior of the components and assemblies.

Designers devote substantial amounts of time and effort to develop the form of objects based on complex aesthetic, cost, manufacturing and other considerations. Much of commercial design activity is based on creating variations around known or existing solutions. A method of automating and extending the creative scope of such activity is developed in this paper.

Parametric design is now widely used in mechanical engineering for embedding geometric relationships in computer aided designs to facilitate alterations and create variations. This paper aims to demonstrate that the capacity of parametric technology, initially developed to handle form variations, can be extended to form generation.
2. Parametric Design

Most engineering design processes already have a high degree of parameterization pre-built into the design and manufacturing process. Though they may not be known as “parametric forms”; steel sections (eg. I sections), nuts, bolts and most engineering components are based on parametric formulation – in that they are based on generic form.

Parametric Technology is now part and parcel of most the 3D engineering CAD programs (that are based on the ACIS Solid Modeling Kernel used in Pro Engineer, Solid Works etc.). It is now very much part of the engineering design process. However, parametric technology is currently used in its elementary form in considering minor variations, costing and tolerance issues in fully configured designs and to keep the design flexible to enable late stream changes without having to re-work the entire design.

3. Other form generation techniques

Currently there are many different types of form generation techniques that are being developed and used for a wide variety of applications, such as computer games, design of structural systems and micro devices. Genetically inspired algorithms are now able to create a wide variety of design configurations and have proven capable of generating novel solutions to clearly definable design problems. Amongst these techniques Genetic Algorithms and simulated annealing are the most well known.

The computational morphing of biological shapes “Bio-Morphs” (Dawkings 1986) has the inspired the development of similar techniques for generating architectural form allowing the user to rank the selection. Frazer (1995) proposed “Evolutionary Architecture” and O’Reilly and Ramachandran (1998) have proposed a Generative Genetic Explorer to create novel 3D Architectural form while Soddu (2002) advocates techniques based on Morphogenesis to generate architectural form. His work aims “...to construct Morphogenesis codes able to generate endless Visionary Variations”.

Other shape generation techniques include “Shape Grammar” (Stinny 1980) based on functional modeling where components and
their functions are logically connected to each other and used as the basis for form generation. A generative design technique combining shape grammar and genetic algorithm has been proposed by Benjamin Loomis (Loomis 2002). Del Coates (2002) has discussed parametric product variations in product design.

Computer life and computer art programs (Todd et al. 1992) make inspiring use of generative techniques. The visual effects of parametrically generated curves have been explored in detail by Mathew et al. (Mathew et al. 2001). Of these techniques; Morphogenesis – the creation of shape, for the biologists it is the process by which living things develop organized structure – is central to the parametric proposition of form development, as it engenders the essence of geometry of form in such away that variations can be created.

4. Proposed method

Proposed here is a method for form generation based on random parametric variations of a generic configuration. It may be structured as an iterative process where a widely varying range of solutions are first generated, and then a few candidate solutions are chosen for further refinement using more conservative parametric variation. The parametric design process may be broken down into 6 steps:

1. **Study of existing solutions** – Parametric from generation is largely based on existing solutions. These solutions contain important information about the design problem and help to set the limits on variations. Design generation in this method is purely a random process. The only source of “knowledge” of the design problem is derived from the geometric logic of existing solutions, hence a wide range of solutions need to be studied and a desirable “solution set” be identified from this.

2. **Creating the Generic Form** – Creating the generic form capable of generating a wide variety of design variations is a critical step in the design process and it may be achieved experimentally by trial and error. It is important to achieve this with a minimal number of CAD primitives (lines, planes,
splines, points etc.) and minimum number of geometric operations (extrusions, sweeps, lofts etc.) controlled by a minimum number of parameters. This is essential to help create significant variations. It is important that the generic design be configured to be geometrically stable – to ensure that parametric variations do not destroy the shape and inherent geometric relationships. It is also important to achieve this without over constraining geometric relationships that could curtail its generative expression. An appropriately configured generic model will yield a good proportion of geometrically viable shapes.

3. **Setting limits & relationships** – the geometric envelope for variations is based on the study of existing solutions. The scope of the exploration is controlled by this design envelope. It is important not to over constrain this envelope as it will reduce the possibility of unexpected yet viable solutions. Hence it is advisable to enlarge the envelope if extraordinary and unexpected propositions are desired. Such limits may be set as maximum and minimum values. They may also be set proportionally (as a constrained randomly varying value) to other design parameters.

4. **Form Generation** – Random numbers are used to generate variations within the geometric envelope. The number of solutions to be generated is chosen according to desired scope of the search and the computational capacity available.

5. **Form Selection** - This can be implemented by the use of expert judgment, consumer preference or by quantifiable criteria. The form generated may be rendered realistically for visual evaluation. As a large number of form propositions is usually generated, it is important to first eliminate similar solutions by visual grouping or by use of numerical methods. The initial selection should identify a number of solutions that are geometrically dissimilar to each other.

6. **Design refinement** – The same design process may be conducted on the selected dissimilar solutions. Some additional variables and features may be added to fine tunes the solution.
The geometric envelope needs to be narrowed down significantly to ensure that the next crop of solutions is close to the pre-selected solution.

The proposed method is structured to ensure a widest possible search in a region close to known solutions. The most promising region of the solution space – identified by studying existing successful and acceptable design solutions – is used here as the starting point for design generation. The geometric envelope regulates the size of the solution space that is explored and needs to be enlarged if novel solutions are desired.

5. Implementation

Previous attempts at design generation have relied heavily on setting up complex geometrical relationships. Parametric CAD packages have now matured to a point where many such relationships can be set without using programming languages. The Genometric method is based on using the embedded parametric and geometric relational capabilities of these parametric CAD systems to maintain the geometric logic of the models.

Some of the generated solutions will have geometric inconsistencies and will fail to generate solutions. This is be expected. It is acceptable as the efforts required to structure the generic form to avoid such errors will be prohibitive and the constraints that may be required to avoid such failure will be restrictive of its generative capacity.

A minimal number of geometric points are chosen to drive the geometry of the basic form as shown in Figure 1 & 2. In this example the radial values of 3 points are chosen to drive the design and define the design envelope. The limits of the geometry are established based on existing solutions. A geometric table is created using random numbers to generate a wide range of values within the chosen range. Simple numerical techniques can also be used to ensure the creation of a spread of values within the given geometric envelope. This table is then used by the CAD program to generate the design propositions.

The complexity of the generic form is controlled carefully to ensure both simplicity and much as possible accurate representation of desired forms. Its complexity of the form is in someway related to the
number of points that are required to represent it. The number of features, points, defining splines, planes and relational equations etc. and the resulting size of the CAD file may give a good indication of the complexity of the form.

6. Example of Parametric Design Generation

Some examples are presented here to demonstrate the application of the parametric design method with various levels of complexity. Figures 1 & 2 illustrate the geometric formation of limits. Table 2 illustrates shapes of wash basins generated parametrically and Figure 4 illustrates shapes of furniture.

The generic geometry of a glass vase is created using splines as shown in Figure 1 & 2. The spline is made of 6 points of which the horizontal dimensions of 3 points are varied randomly between the limits set in Figure 1 & 2. The resulting parametric variations are shown in Table 1.

Figure 1. Maximum Geometric Envelope Figure 2. Minimum Geometric Envelope
The shapes shown below in Table 1 have been generated using random numbers within the geometric envelope shown in Figure 1 & 2.

TABLE 1. Parametric Glass Vases
Some similarity of shapes may be observed amongst this wide variety of shapes. The next step would be the selection of some distinctive shapes and creating more detailed variations on them.

TABLE 2. Parametric Wash Tubs
7. Applications of Parametric form Design

Parametric form generation may be used for a wide variety of complex form design problems. It is especially useful in the concept generation phase of the design process where it is beneficial to consider a large number of possible solutions before proceeding to the selection and refinement phase. Often, in engineering designs, the relationship between form and specific engineering properties is well understood and linked by an analytical relationship. But the multiplicity of the parameters involved and the complexity caused by their interrelationship and difficulties resulting from the inherent complexities of engineering design problems makes most engineering design problem difficult to resolve using simple procedural techniques. However, procedural design techniques that rely on analytical relationships are extensively used for solving well understood engineering design problems with a limited number of variables of limited complexity. Once the design exceeds a certain complexity this may not be possible (with the given computational capacities) after which point engineers rely on compiled knowledge or experiential knowledge to guide their design process. In areas of form design activity where such analytical link is non-existent, or sufficiently complex the parametric form design becomes a useful design tool for generating form design propositions.

Designers often work in an environment in which they are discouraged from practicing truly creative design due to marketing, branding, manufacturing and other constraints – requiring the designers to perform what is considered to be “routine design”. While there are well developed formal methods which encourage designers to consider all possible solutions, there are few methods available for
the practice of “routine design” which constitutes in practice the bulk of design activity. Parametric form design is eminently suitable for such applications.

Parametric Design is now widely used in engineering design where the shaping of form plays an important role in structural and mechanical and thermal design. For the most part engineers understand how structural, mechanical & thermal properties of components and assemblies vary with geometry. They also understand the limits and regions in which physically properties change predictably with form. As the parametric method rely on geometry to define design possibilities, it is only suitable for form designs in which the performance or merit of the design vary gradually and continuously with geometric changes of the form. It is not suitable for the rare design problems in which small geometric changes in form result in significant change in design outcome (e.g. aerodynamic design in which small geometric changes can cause the flow to change from laminar to turbulent flow).

In the consumer product industry designers are encouraged to come up with highly creative designs within tightly constrained brand identity which is often defined by geometric form. Parametric form design allows the designer to stay close to the brand identity by structuring the generic form to encapsulate the important geometric aspects of the brand.

8. Creative Limits

The Genometric form design method is limited to the exploration of design possibilities around existing solutions. Initially this may appear to limit the creative scope of solutions that are generated. This was found not to be so. It was found that “errors” in modeling could lead to unexpected results producing unexpected and perhaps “creative” results. The biological analogy for this may be mutation caused by errors in copying the genetic code - without such errors mutation and therefore evolution will not be possible. The other effect is similar to what is observed in biology as “genetic drift” – whereby small changes iterated over long periods can produce significantly different forms to the original form. Hence if the parametric design method is applied iteratively it too could generate quite rich variety of form propositions – which we find in biological life forms.
Two examples are shown in which the genometric method has produced some unexpected yet viable designs – which may be termed creative solutions depending on the definition of creativity. Shown in Figure 4 is a revolute form created by changing one parameter – the revolve angle. Here the parametric design method has led to unexpected design solutions generating a wide range of possibilities.

Figure 4. Creative Furniture Design

Another example is from wash basin design as shown in Figure 7 where the variations led to a new and unexpected yet functional form that combines the conventional wash basin shown in Figure 5 with the table shown in Figure 6.

Figure 5. Generic Wash Basin  Figure 6. Inset Wash Basin
9. The case for Reverse Design

Much of the engineering and product design activities takes place in industrial and commercial environments where the design outcomes are fairly predictable and design possibilities are explored within a well-constrained region of possibilities frequently as a result of business, management and marketing decisions. Designers are often not in a position to alter these priorities neither are they expected to change them. They are often expected to come up with design possibilities close to well known and successful design solutions, but at the same time sufficiently different from existing products for branding and marketing purposes.

The art of reverse engineering is now widely and profitably practiced as an alternative to conventional design and product development practice. It’s success and wide spread use has conferred it respectability and recognition as a formal product development method. A reverse engineering based approach for product design is widely practice has also been proposed as a formal method (Shih-Wen et al. 2003) Genometric form generation may be seen as a “reverse design” process sharing some of the advantages and benefits of reverse engineering, especially in reducing the development time, reducing the risk and cost involved in new product development.
Historically, in the practice of design only a very small portion of projects involve truly creative design. Creative design entails significant risk, cost, investment and expertise. Moreover, creative design efforts do not necessarily ensure successful outcomes. It is also known that creative design solutions, apart from being rare, mostly accomplished without the use of formal design methods. In this context, it is import to examine the value of using conventional design methods for routine design, as they rely on first understanding, then specifying, generating and evaluating possible – all of which may be not necessary for generating acceptable design outcomes.

Whatever design approach is used, it is observed that most design outcomes are similar to existing design outcomes that are readily available for the designer’s scrutiny at the very beginning of the design process. The geometry of these solutions embodies information and knowledge relevant to the design problem. The parametric form design method uses this information directly to generate design solutions.

For design problems for which many design solutions already exists it may be appropriate to direct design and evaluation resources around such solutions (or solution space) rather than to dispense such efforts on the widest possible search space as conventional design processes would encourage. The intent to limit design exploration to a smaller search space could greatly improve the quality of solution as greater efforts could be mobilized to explore realistic solutions in greater detail rather than dispense such efforts over a larger potentially unviable area.

10. Further Research

The creative scope and depth of application of the parametric form design method needs to be established in more detail. This may be achieved by conducting the equivalent of Turing test, by comparing forms of existing products to forms generated by the parametric form design method. Such a test may help establish the perceived level of “creativity” of the method which may be achieved by novices as well as expert designers.

Methods of setting up the initial parametric forms need to be further explored and developed. Currently, this is being achieved experimentally. It is observed that a smaller number of variables
create a wider range of solutions than a higher number of variables for a given number of propositions. This relationship between the number of variables and form variations needs to be better understood. The implications of setting up the design envelopes in an absolute sense and relativistic sense need to be better understood.

It is clear that the methods of selecting solutions and the setting up of design envelopes need to be further developed. Image processing techniques may be used as a way to accomplish this. Computational methods for selecting candidate solutions that are sufficiently dissimilar to each other also need to be developed. Texture and colour can be added as parametric elements to greatly enhance the capacity of this method.

As form design methodology is a search for possible geometric configurations, design processes aim to ensure thoroughness and efficiency. The quality of the design solutions is governed by the thoroughness of the search process - which is related to size of the search space in which solutions are searched – and the efficiency with which this often enormous search space is explored. In turn, this often depends very much on the search strategy employed. These issues need to be addressed from geometric and computational point of view.

11. Conclusion

This paper establishes the theoretical basis and demonstrates the viability of the Genometric form design methodology. It outline the basic steps required to implement such a design process. A case is made for a reverse design process for product form design. Issues on managing the creative scope of the design exploration are discussed. It argued that the Genometric method could yield form design solutions that may be considered to be “creative”. This needs to be established experimentally. The Genometric design method frees the designer to focus more on choice, evaluation and refinement rather than
generation – which is entirely left to the computer. This paper demonstrates that it is possible.

References


Soddu, C: 2002, *Visionary Variations* - Exhibition Visual Art Centre, Hong Kong Museum, Hong Kong Park, 7A Kennedy Road, Central, HK


HERMES: A COMPUTATIONAL TOOL FOR PROPORTIONALITY ANALYSIS IN DESIGN

Abstract. A computational tool for proportionality analysis in architectural composition is introduced. The morphological structure of a design is represented with a list of dimensions associated with design components and every possible triplet combinations of the list is computed and tested for its fit to one of the 11 possible proportionality sets. Using factor analysis of the contributions of the different proportionalities to the design, a possible morphological resemblance of different design artifacts is proposed. The case studies of this work are the 38 plans from Palladio’s second book of architecture.

1. Introduction

The absence of computational tools for the analysis of proportion of designs has been a persistent problem in the field of formal composition in architectural design. Analysis of existing designs still requires an enormous amount of patience and persistence from the researcher to undertake the proportional study with pencil and paper. Synthesis of new designs with proportional qualities requires command of this body of knowledge that very few architects nowadays possess and of course again an enormous amount of patience to get it right.

In this paper a computational tool, Hermes, is introduced as a tool for proportionality analysis based on the theory of means. In order to link this project to the long list of research projects that have tackled proportion, the corpus of the designs has been selected from Palladio’s buildings to test the results of the algorithms versus the existing literature. 38 plans of Palladio’s designs have been studied and the results of the analysis are explained with statistical representations using factor analysis, a multivariate data reduction technique. In the process, the emerging morphological resemblances between particular plans are tested to the historical references or theories already established (Mitrovic, 1990; Wittkower, 1971; Shin, 1996; Hersey and Freedman, 1992; March 1999; Howard and Longair, M: 1982, Tavernor and Schofield, 1997)

2. Ratio, Proportion, and Proportionality

A ratio is a relation between two numbers and proportion is a relation between two ratios. The least set of numbers that can establish a proportion is 3. For 3 numbers $a$, $b$, $c$, and $a<b<c$, there are 3 possible outcomes of
comparisons, 1 unique case of equality, $a:b = b:c$, and 2 cases of inequality, $a:b < b:c$ and $a:b > b:c$. For each case of inequality, there can be an infinite number of sub-cases with respect to the actual magnitudes or multitudes involved to the comparison. Among these relationships, some are more significant than others; for example, for three numbers $a, b, c$, and $a < b < c$, if $(1/c) - (1/b) = (1/b) - (1/a)$, the inequality $a:b < b:c$ can be rewritten as an equality, namely, $(c-b)/c = (b-a)/a$. The problem has been nicely solved in antiquity by Greek mathematicians in a series of successive attempts, initially proposing 2 more such equalities by Archytas, the arithmetic and the harmonic mean, latter 3 more, possibly by Eudoxus, and finally 2 additional distinct sets of 4 by Nicomachus and Pappus respectively, with 3 overlapping cases among them, bringing the total number of inequalities to 10. These 10 relationships of ratios plus the first initial relation of equality, the geometric mean, brought the number of comparisons to 11 and they were all treated informingly under the heading of proportionality or theory of means (Heath, 1921). Among these 11 ways of comparing two ratios involving 3 magnitudes or multitudes, only 4 survive in current discourse, the initial three, the geometric, arithmetic, and harmonic mean, and the extreme and mean ratio, otherwise currently known as the golden mean (Herz-Fischler, 1987).

3. Proportionality Analysis

The proportionality analysis used in this project has three principal components. The first is the collection of a set of numbers from the dimension of plans illustrated in the second book of the *Quattro libri*. From the total of 49 buildings, the dimensions of 38 plans of villas and palaces are selected from the dataset provided in “Harmonic Proportion and Palladio’s Quattro Libri” (Howard and Longair 1982). The set of numbers is listed in a text file for the input of the analysis.

The second is the computation and categorization of possible proportionalities embedded in the triplet combinations within the input dataset of individual buildings. In this process, a computation tool for the analysis titled “Hermes,” is introduced. It is programmed in Autolisp, a dialect of Lisp, and it is designed as a plug-in application for Autocad, an existing design software. Figure 1 illustrates the structure of Hermes in a flowchart diagram. Hermes computes every possible triplet combination and generates a spreadsheet that shows the relationship of specific proportional triplets over the total number of triplet combinations of measurements. Other statistical relationships are computed as well (Penev, and Atick 1996). Table 1 shows the percentage of the contribution of individual proportionality to the 38 plans in the second book of the Quattro libri.
The third is the clustering of the 38 different plans of Palladio according to their similar pattern in proportionalities with factor analysis, one of multivariate statistical analysis techniques. The similarities among the plans on a particular factor are represented as a factor score. (Johnson and Dean 1998:550-557, Kachigan 1991:244). A correlation of architectural components in the plan with actual triplet number combinations is attempted as well to link proportionality analysis to architectural composition. Figure 2 shows a visual comparison between the proportionality structure of Rotonda and villa Trissino.

4. Discussion
In this research project, Hermes, a computational tool for proportionality analysis in architectural design has been introduced. All possible triplet combinations of input dimensions extracted from a plan have been computed and have been used to classify designs according to a statistical and quantitative analysis. With a multivariate data reduction technique called factor analysis, a possible resemblance of proportionality structure between different plans has been studied.
Figure 1. The Structure of Hermes
TABLE 1. The percentage of the contribution of individual proportionality to 38 plans in the second book of the Quattro libri

<table>
<thead>
<tr>
<th>name</th>
<th>prop1</th>
<th>prop2</th>
<th>prop3</th>
<th>prop4</th>
<th>prop5</th>
<th>prop6</th>
<th>prop7</th>
<th>prop8</th>
<th>prop9</th>
<th>prop10</th>
<th>prop11</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLPl'Abboni</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HLPl'Bagnato</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HLPl'Cappelli</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HLPl'Cappelli</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HLPl'Cappelli</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HLPl'Cappelli</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HLPl'Cappelli</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HLPl'Cappelli</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HLPl'Cappelli</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HLPl'Cappelli</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HLPl'Cappelli</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HLPl'Cappelli</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Figure 2. The visual comparison between the proportionality structure of Rotonda and villa Trissino
References

Workshop 3: Schedule

Implementation Issues in Generative Design Systems
Saturday 17 July 2004, 14:00 - 17:45

14:00: Introductory note
Chairs

14:10: Session 1: Strict grammar-oriented approach
Moderator: José Duarte, Technical University of Lisbon, Portugal

A set-based shape grammar interpreter, with thoughts on emergence (15 + 5 min)
Andrew I-Kang Li and Lau Man Kuen, The Chinese University of Hong Kong, China

Grammars, Sorts and Implementation (15 + 5 min)
Rudi Stouffs and Ramesh Krishnamurti, CMU, USA

Shaper 2D: an interactive shape grammar interpreter (15 + 5 min)
Miranda McGill, MIT, USA

A studio exercise in rule-based composition (15 + 5 min)
Sotirios Kotsopoulos and Haldane Liew, MIT, USA

Universal design + case driven computation (15 + 5 min)
Stylianos Dritsas, MIT, USA

15:50 Coffee Break and Poster Presentations

16:00 Session 2: Other approaches
Moderator: Luisa Caldas, Technical University of Lisbon, Portugal

Generative Design: from algorithms to intuitive and interactive software demonstrators (15 + 5 min)
Kristina Shea and Marina Gourtovaia, Cambridge University, UK

Emergent Designer: An Integrated Research and Design Support Tool Based on Models of Complex Systems (15 + 5 min)
Rafal Kicinger, Tomasz Arciszewski, George Mason University, USA

Geometry as a substitute for structural analysis in generative design tools (15 + 5 min)
Martin Hemberg, Una-May O'Reilly, Imperial College, UK and MIT, USA

Arergenia, a mother tongue in infinite variations: identity rises from a soundless site, because birds have no tears (15 + 5 min)
Celestino Soddu and Enrica Colabella, Politecnico di Milano University, Italy

Hermes: a computational tool for proportionality analysis in design (15 + 5 min)
Athanassios Economou, Georgia Tech, USA

17:20 Final Discussion (25 min)

17:45 Closure

Note: Each presentation slot includes paper presentation and software demo (15 min), and discussion (5 min). Presenters without software demos will have their presentations shortened to 8 min.