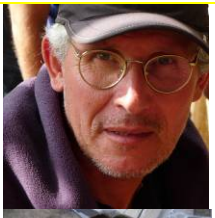


Programming generative grammars (paper)

Topic: Art, Architecture

Author(s):
Umberto Roncoroni
Lima, Perú, Universidad de Lima, Facultad de Comunicación

Veronica Crousse
Lima, Perú, Pontificia Universidad Católica del Perú, Facultad de Arte

The paper is a summary of a research project report that won the 2014/2015 Multidisciplinary Research Prize granted by the Consorcio de Universidades of Lima, Perù. The goal of the project was to develop new generative programming techniques for string-substitution rewriting systems, such as Shape Grammars and L-Systems, applied to art, design and architecture.
These systems are widely used for their unique creative processes and capabilities to generate complex and emergent forms such as fractals or organic 2D/3D patterns.
Many implementations of such algorithms are available, but they basically work on already known principles and concepts. Much more can be done to develop their efficiency and creative capabilities. The goal is to add more flexibility and interaction to their recursive functions and to achieve truly generative processes; so far, emergence and complexity are only simulations, tricks, or even cheats concocted with pseudo random functions.
Through the procedural analysis of Peruvian pre Columbian designs and software experimentation we have discovered that Shape Grammars, L-Systems and recursive functions do offer unexploited programming capabilities that power up their creative and generative qualities. In this sense, the study of ancient Peruvian design (*tokapus*, q*uipus*, etc.) has improved the finesse of the generative digital process, because this kind of art shares a lot with actual generative design paradigms, such as the connection with nature, complexity, dynamic systems rules, and more.
Speaking of results, we have added to standard recursive algorithms dynamic and real time rules, masks, loops, go to, symbol's waiting lists (like in multithread programming), automatic formulas, calculation, nested L-Systems and grammars, chemical and genetic reactions, auto editing and new interface concepts. The interface design is really important, because the symbolic manipulation, to be truly generative and emergent, must be transparent, editable and fully interactive in real time. In this paper we will show the methodology of our research: first the analytical and visual comparison between natural and pre-Columbian forms, in the second place hand drawn algorithmic designs, finally, the implementation of the new set of recursive rules in a new L-Systems application written in VB 6.0 and C#.

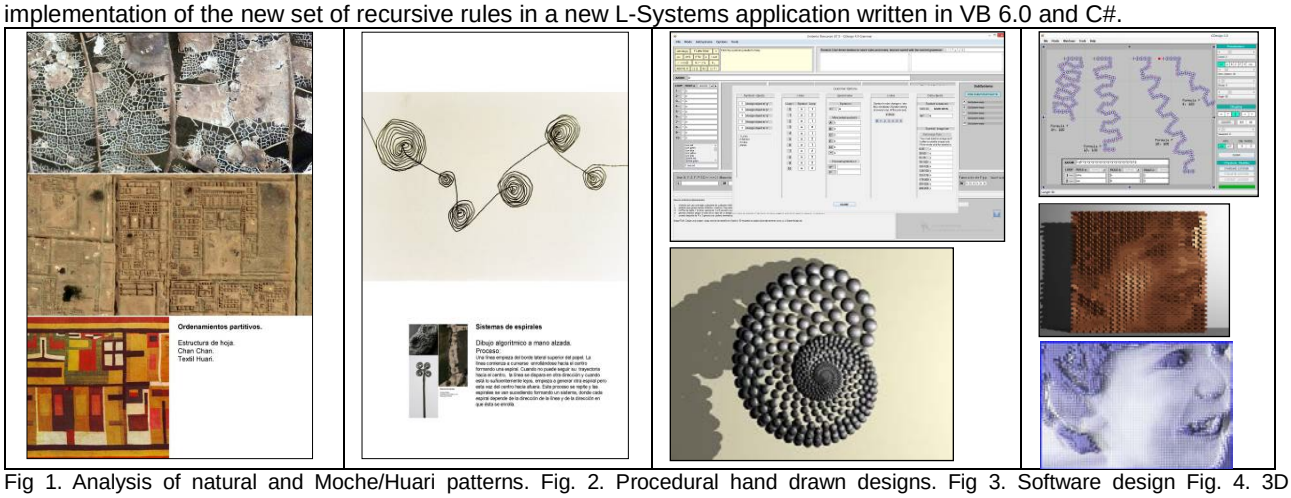


Fig 1. Analysis of natural and Moche/Huari patterns. Fig. 2. Procedural hand drawn designs. Fig 3. Software design Fig. 4. 3D programable double spiral rule with L-System. Fig 5 y 6. Implementation of programmable L-Systems rules.Fig. 7, 8. 3D image processing applications with programmable L-Systems.

hroncoro@ulima.edu.pe
vcrouss@pucp.edu.pe

Key words: art, complexity, nature, pre Columbian culture, Shape Grammars

Main References:

[1] DE PRADA, M. *Arte y naturaleza. El sentido de la irregularidad en el arte y en la arquitectura*. Buenos Aires: Nobuko. (2009).
[2] LEYGHTON, M. *A Generative Theory of Shape.* Berlín: Springer-Verlag. (2001).
[3] CROUSSE, V. *Reencontrado la espacialidad en el arte público del Perú*. Tesis doctoral en Espacio público y regeneración urbana: arte, teoría y conservación del patrimonio. Barcelona: Universidad de Barceloan, Facultad de Bellas Artes. (2011). [en línea]. <http://www.tdx.cat/handle/10803/1551> [Fecha de consulta: 20/8/2015].
[4] ALLASMAA, J. *La mano que piensa. Sabiduría existencial y corporal en la arquitectura*. Barcelona: Ed. Gustavo Gili. (2012).

Programming shape grammars and string substitution rewriting systems for generative art

Dr. Umberto Roncoroni Osio.
Universidad de Lima, Lima, Perù.
www.digitalpoiesis.org
e-mail: hroncoro@ulima.edu.pe

Dra Veronica Crousse Rastelli
PUCP, Lima, Perù.
e-mail: vcrouss@pucp.edu.pe

Abstract
The paper is a summary of a research project report that won the 2014/2015 Multidisciplinary Research Prize granted by the Consorcio de Universidades of Lima, Perù. The goal of the project was to develop new generative programming techniques for string-substitution rewriting systems, such as Shape Grammars and L-Systems, applied to art, design and architecture.
These systems are widely used for their unique creative processes and capabilities to generate complex and emergent forms such as fractals or organic 2D/3D patterns.
Many implementations of such algorithms are available, but they basically work on already known principles and concepts. Much more can be done to develop their efficiency and creative capabilities. The goal is to add more flexibility and interaction to their recursive functions and to achieve truly generative processes; so far, emergence and complexity are only simulations, tricks, or even cheats concocted with pseudo random functions.
Through the procedural analysis of Peruvian pre Columbian designs and software experimentation we have discovered that Shape Grammars, L-Systems and recursive functions do offer unexploited programming capabilities that power up their creative and generative qualities. In this sense, the study of ancient Peruvian design (*tokapus*, q*uipus*, etc.) has improved the finesse of the generative digital process, because this kind of art shares a lot with actual generative design paradigms, such as the connection with nature, complexity, dynamic systems rules, and more.
Speaking of results, we have added to standard recursive algorithms dynamic and real time rules, masks, loops, go to, symbol's waiting lists (like in multithread programming), automatic formulas, calculation, nested L-Systems and grammars, chemical and genetic reactions, auto editing and new interface concepts. The interface design is really important, because the symbolic manipulation, to be truly generative and emergent, must be transparent, editable and fully interactive in real time.
In this paper we will show the methodology of our research: first the analytical and visual comparison between natural and pre-Columbian forms, in the second place hand drawn algorithmic designs, finally, the implementation of the new set of recursive rules in a new L-Systems application written in VB 6.0 and C#.

A short introduction to shape grammars and LSystems

Shape grammars where first introduced by Stiny and Gips [1] in the seventies, following Chomsky's generative linguistic paradigm. L-Systems, a shape grammars subgenre,

where developed by biologist Lindenmayer [2][3] to study and simulate by computational means the morphogenesis and morphology of plants.
Both systems are String-Substitution Rewriting Systems (SSRS), which means that they can build procedurally complex forms with a set of alphanumeric symbols, a substitution process that transform these symbols recursively and a set of rules that specify the transformation's algorithm and parameters.

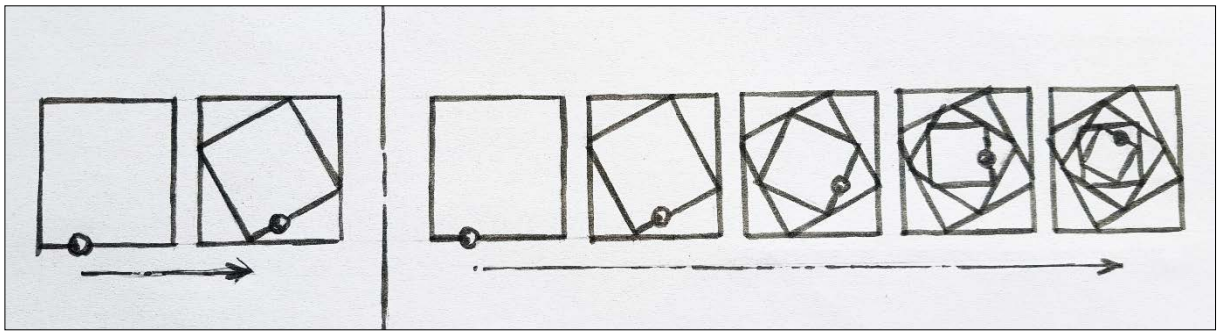


Fig.1. From left to to right: initiator, the rule and 5 steps of the SSRS development.

SSRS are highly appreciated in art, industrial design and architetcure for their fractal capabilities, simplicity and efficiency. They can build very complex forms by means of simple rules and a even more simple algorithms [4].
With the help of controlled randomness and proper parametrization, SSRS can simulate complex irregular natural forms, like rocks, mountains and organic forms like trees [5].

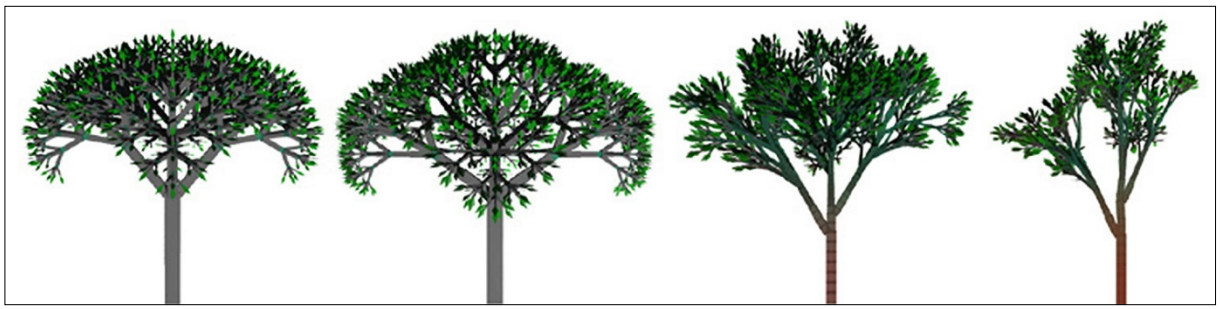


Fig.2. Increasing randomness in a L-Systems tree grammar.

Nevertheless, SSRS are still underestimated and its capabilities unexploited. They can give a lot more than actual implementations allow to do. The goal of this paper is to develop techniques to improve the scope and the possibilities of these recursive systems. In fact, they could be considered a complete programming language.
Our SSRS research and development was grounded on the analysis of natural forms [6][7][8][9] and pre Columbian designs [10]. Both formal contexts share procedural, recursive and generative rules; this procedural correspondence unlocked new insights on emergent algorithms. We cannot develop this aspect of our research, but the following images hopefully will offer some evidences on the matter.

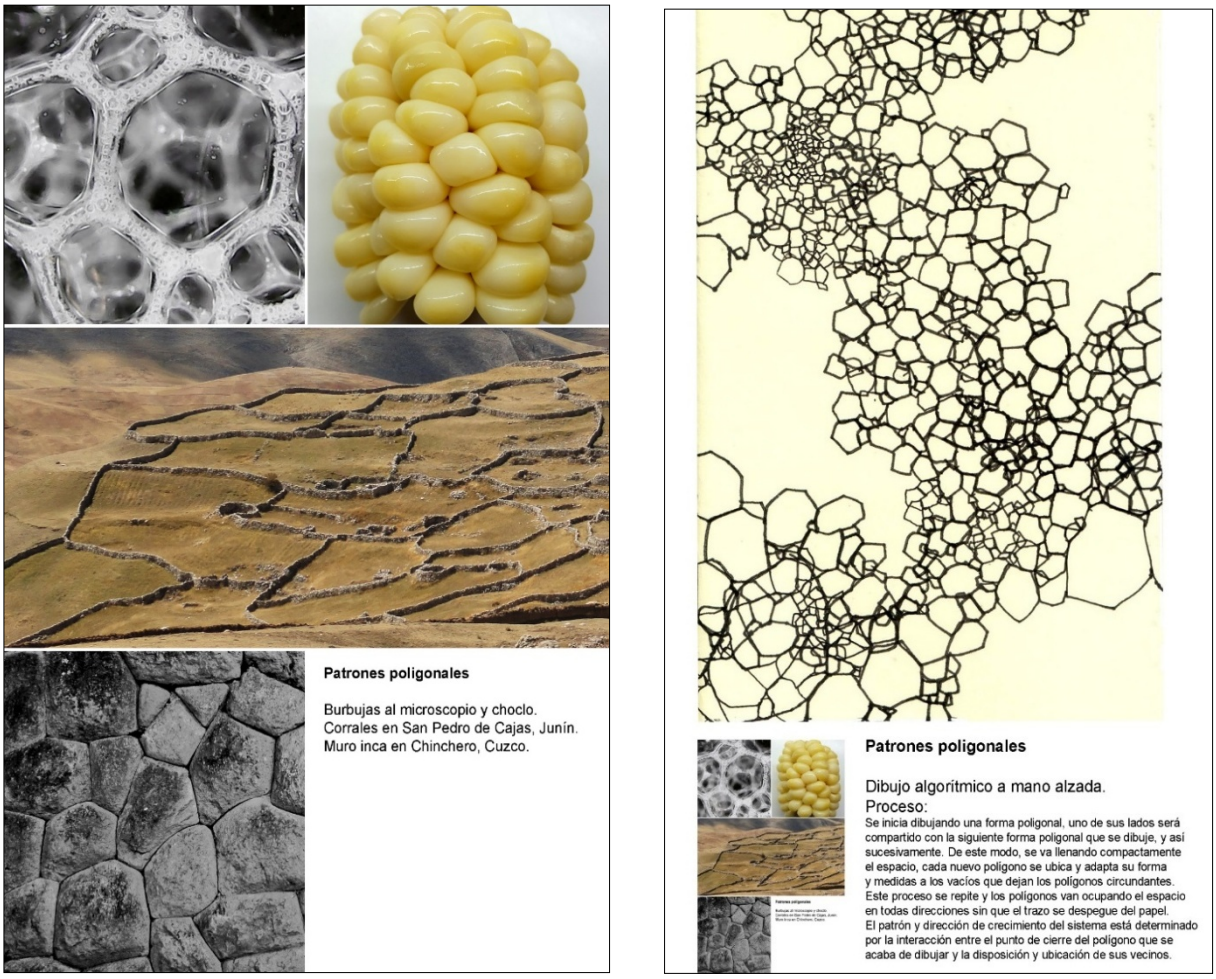


Fig.3. Analogies between natural and pre Columbian patterns. These analogies are not casualties, ancient Peruvian architectures were built in accordance with cosmic and religious principles and a deep understanding of the natural environment. The image on the left shows the artistic research of the formal procedures implicit in nature and Peruvian art. Plates by Veronica Crousse.

Following this hypothesis, we will begin analyzing the problems of shape grammars and L-Systems, then we will show some strategies to bypass these limitations, finally we will introduce some new techniques and explain how they could be applied and implemented. We will use VisualStudio C# and the L-Systems application GDesign [3].

Last but not least, I would like to express gratitude to the Consorcio de Universidades of Lima, a joint venture between Universidad de Lima, PUCP, Cayetano Heredia and Pacifico universities, which supported during 2015 our interdisciplinary research.

Shape Grammars and L-Systems, possibilities and limitations

SSRS are very powerful machines to build complex and fractal forms thanks to the recursive and parametric nature of its processes and the simplicity of its alphanumeric symbolic language [11]. The recursive substitution process generates complex forms in few steps with very simple rules. Besides, these systems have been widely reengineered, and its basic algorithms have been improved using physical computing, genetic programming, artificial life and more, resulting into new implementations of the SSRS process like parametric systems, timed systems, context systems and more [12]. So far, SSRS works very well with fractal forms, but the fractal process is deterministic and, by a generative point of view, rigid and mathematically defined and the kernel of the substitution process is always the same [13]. The fact is that the substitution process is lineal, scarcely

interactive and rigid.
For instance, there are many complex forms that can't be obtained by standard, parametric or context shape grammars or other recursive substitution techniques, as the pre Columbian forms analysis clearly demonstrated. As an evidence, we can consider the case of emergence and organic forms, whose diversity is usually faked using pseudo random number generators. The apparent complexity of computational forms is interpreted as emergent, making a confusion between complexity and complication that are not exactly the same. So SSRS claims for a generative nobility that they are not truly entitled for.

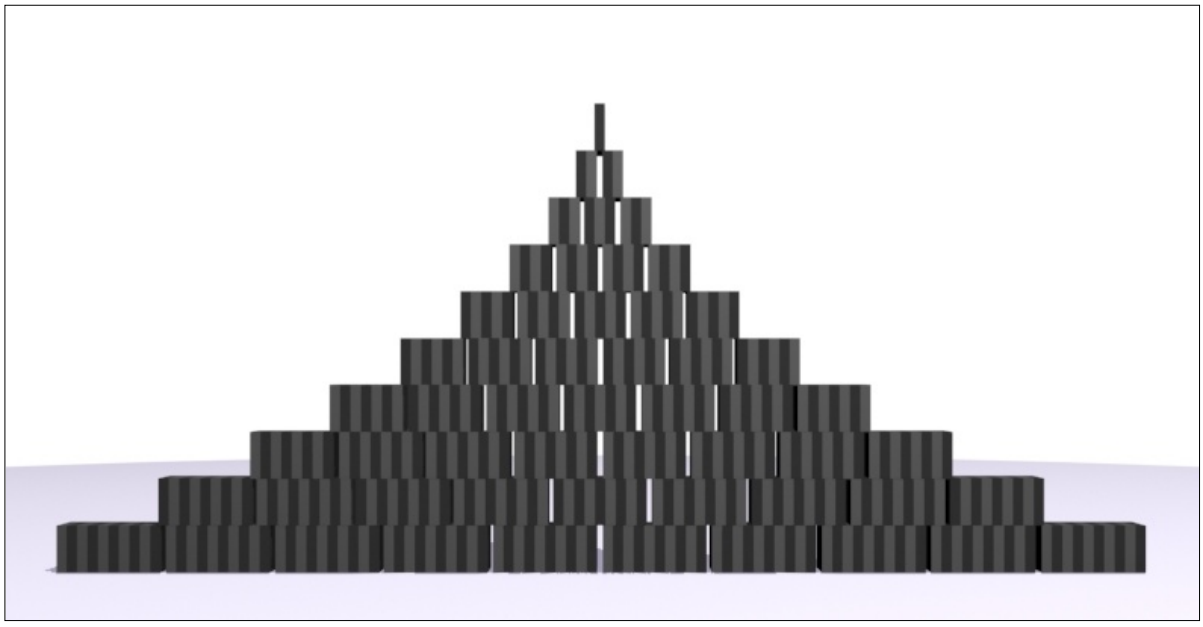


Fig.5. A complex branched structure where every branch holds the number of elements of its branch degree. The substitution rule provides 1 branch with 1 element, 2 branches with 2 elements, 3 branches with 3 elements, etc. This is impossible to construct with usual SSRS rules because the rule must change at every level.

The problem is not just the lack of interactivity of the recursive substitution process as such, but also in its software implementations, including software architecture and the interface design. This is very important because the development of the recursive process eventually converts the process itself into something impossible to understand and visualize. To fix this limitations there are a lot of possibilities:

- Expand the vocabulary of a given SSRS, introducing new symbols for specific tasks or programming needs
- Develop new kind of rules with automatic context sensibility or randomness management
- Improve the control over the SSRS workflow
- Design original interface patterns
- Create new tools for editing and postproduction of SSRS objects

In the following chapter we will explain in detail how these techniques can be used to transform a SSRS implementation into a complete programming system[4].

[4] Download the LSystem application from our web page http://www.digitalpoiesis.org

Programming string-substitution rewriting systems

So the task is to investigate new symbols, rules, interface designs, process flow controls and editing tools using L-Systems as our framework. It is important to stress that we found these algorithms and techniques analyzing natural and pre Columbian patterns, and we tested them during our Digital Architecture workshops.

Anyway, the basic principle is: the more you create links between parts of a shape grammar and its symbols, the more generative the process will be.

Expansion of the symbolic language of L-Systems

1) The most important new symbol that we introduced in the vocabulary of L-Systems is the "=" symbol. "=" freezes its predecessor symbol for 1 turn. "=" can be stacked to multiply its effect, so: "aa===" means that the first "a" will normally produce its rule, but the second "a" will be inactive for three iterations. This simple idea do offer a great control on the substitution process. Many structures are impossible to achieve without "=".

2) Automatic symbols "q" and "g". These symbols automatically generate 2, 3 or n copies of itself in every iteration. For example, if we have a "q" in the axiom, the first iteration will produce "qq", the second iteration "qqq", etc. "q" and "g" can be associated with "=" to block the production for 1 or more iterations. Using symbol "¡" can invert "q, g" to decrease the symbol production. If we have string "qqqqqq¡", the the next iteration will produce "qqqqq¡", then "qqqq¡" etc, and stops when the last "q" is deleted.

3) "{,}".Create patterns of symbols within a rule. Easy to edit with an integer to set how many patterns will be generated. It is useful to squeeze a long set of symbols to facilitate the writing and editing of rules. The number can be set as a parameter. Example:

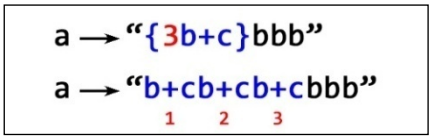


Fig. 6. "{}" example

4) "s". Recursive symbol "s". This symbol takes the complete SSRS string developed in the current iteration. "s" allows sub recursion, say a recursion inside the main recursion process.

5) "n". "n" produces as many symbols as its level of occurrence in the current iteration. See the following example:

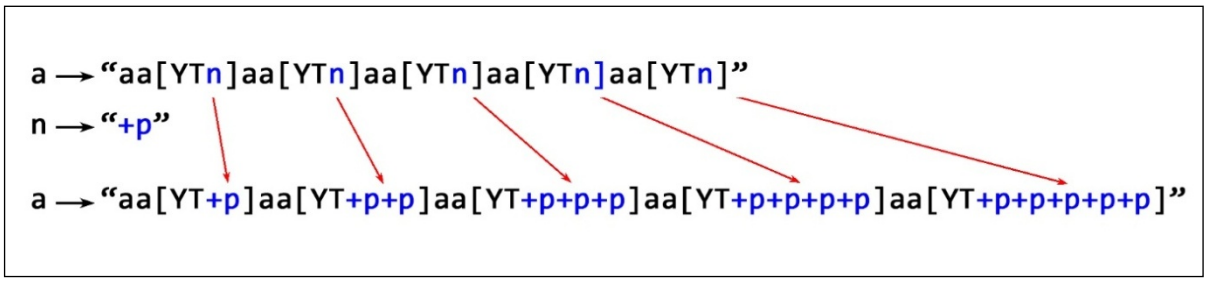


Fig. 7. "n" example. ("YT+" means a Z rotation in the Y axis).

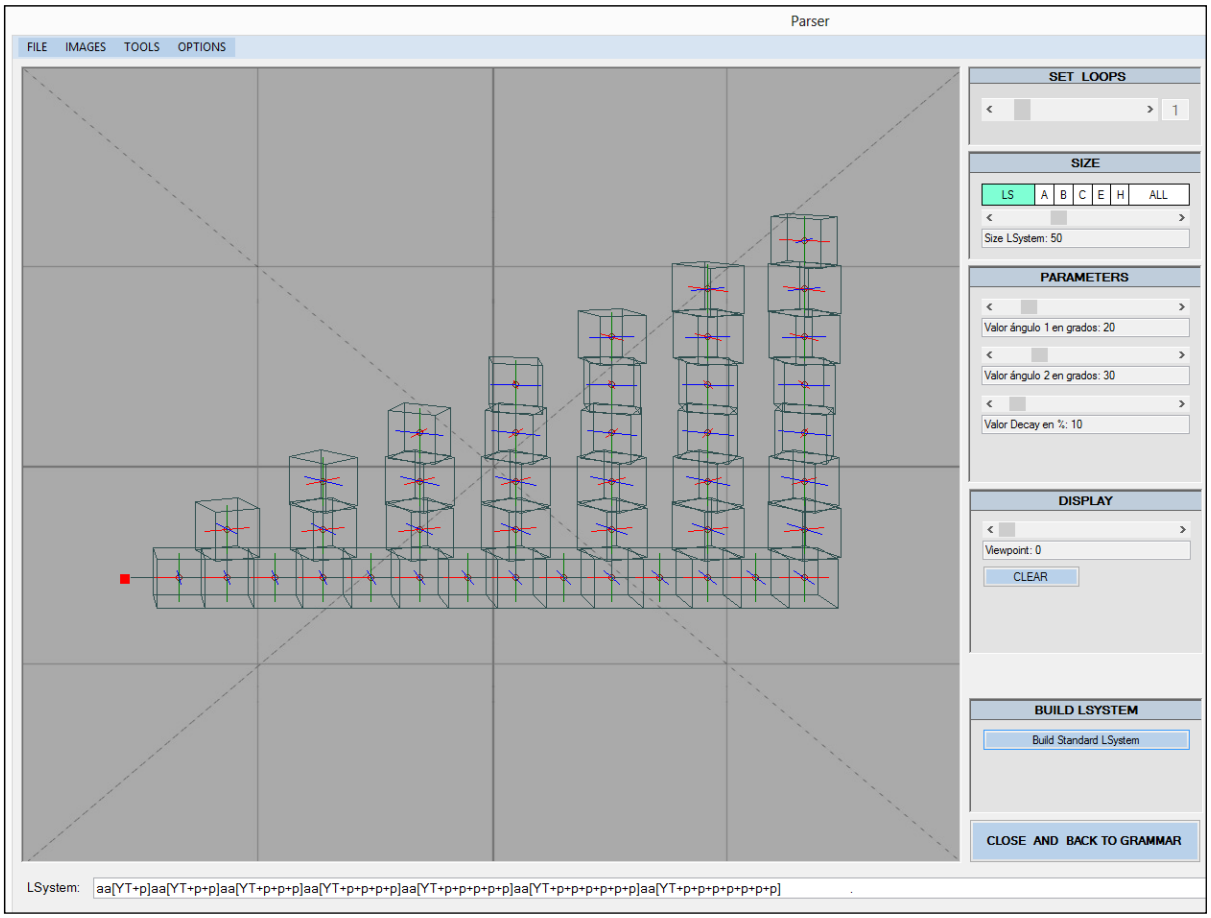


Fig. 8. “n” example model. Here “n” commands the number of vertical elements and their rotation angles.

Additionally, “n” has different production rules for every iteration step and it can be associated with a context symbol “ñ”. The rule of “ñ” depends of its current “n” value. This allows complex iterations and the automatic connection of rules. Consider the following grammar:

```
axiom = “D+a+a+a”

a → “a[P<n]”    n → “a[T<ñ]”    ñ → “a”

1 → D+a[P<n]+a[P<n]+a[P<n]+a[P<n]

2 → D+a[P<a[T<ñ]] +a[P<a[T<ñ]a[T<ñ]] +a[P<a[T<ñ]a[T<ñ]a[T<ñ]

3 → D+a[P<a[T<a]]
    +a[P<a[T<aa]a[T<aa]]
    +a[P<a[T<aaa]a[T<aaa]a[T<aaa]
```

Fig. 9. “n” and “ñ” example grammar. The first “n” produces “a[T<ñ]”, the second “n” “a[T<ñ] a[T<ñ]”, the third “n” “a[T<ñ] a[T<ñ] a[T<ñ]”, marked in blue. Then first “ñ” (of the first “n”) produces “a”, because it is the first, the second “ñ” produces “aa”, because was generated by the second “n”, and so on. So “ñ” is automatically connected to “n” which is automatically connected with “a”.

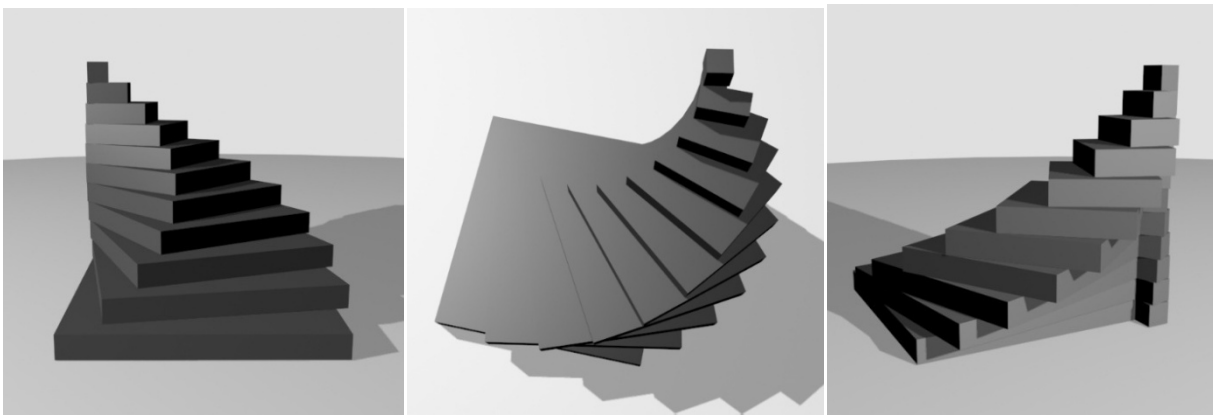


Fig. 10. Connection of dimensions and rotations using “n” and “ñ”.

6) “?”. Symbol “?” apply a trigonometric formula or a set of data to rotation angles. In our implementation formula and data can be set using interactive interface elements.

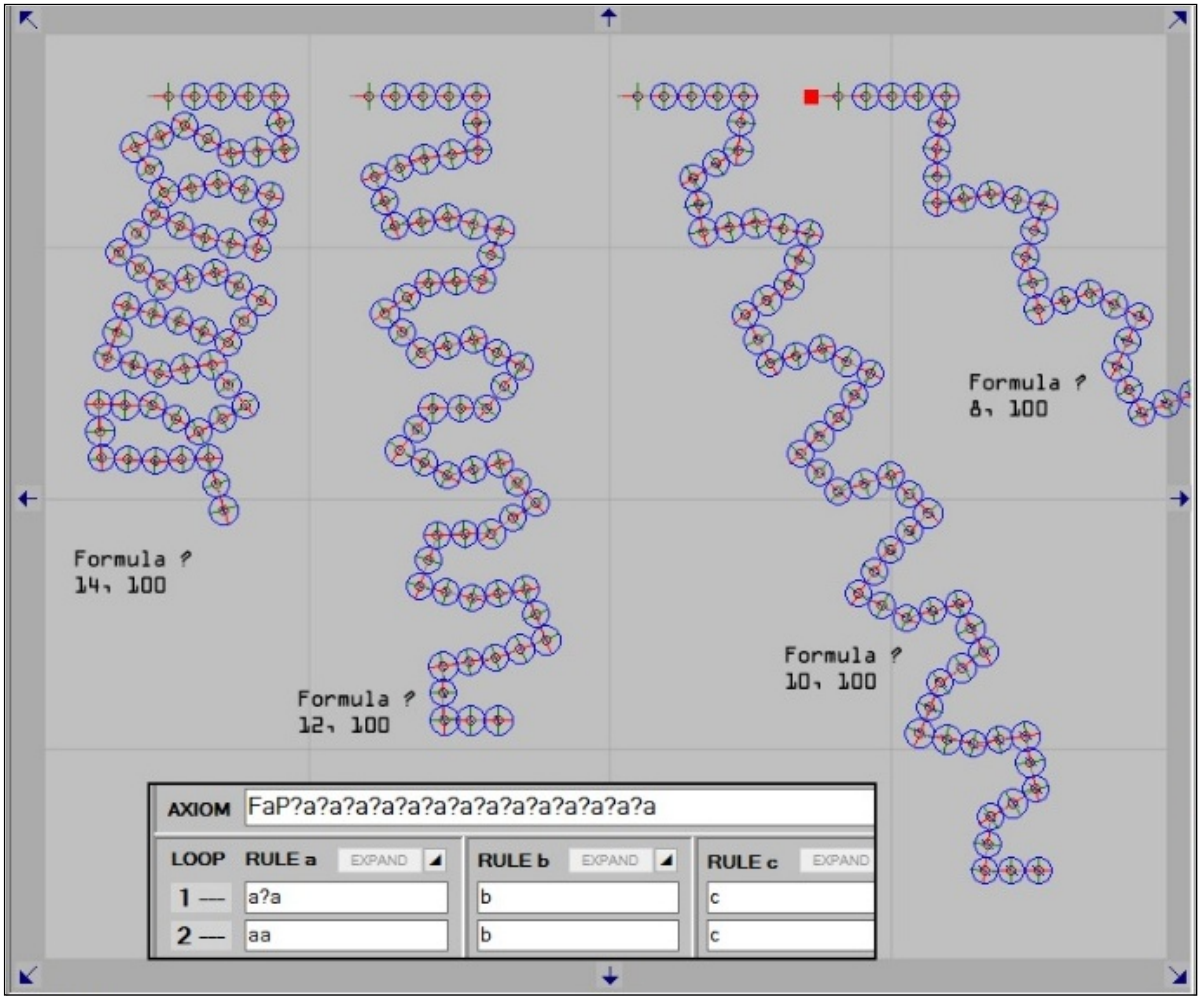


Fig. 11. Example of “?” symbol. User can set the mathematic expression and its variables values.

Functions and procedures

L.Systems can be improved with new symbols and with special rules and functions. In the following chapter we will show some solutions for random string generation, functions, and rules connected with external data source like databases or images.

1) Random rules and functions. Randomness is very important to achieve difference and variety. The problem is how to control random number generation, for instance, to maintain

the coherence of random values between parts or branches of the model. We implemented symbols “R,S,W” as macros or functions where you can set the min/max range of string’s length and the symbols to be randomly inserted. User can choose between random length with one unique symbol, fixed length with random symbols, or both. “R,S,W” work with “=” symbol, so you can generate a random value at the beginning of the process and maintain this value later on. For instance: “RbbaP+R=” means that “R=” uses the random value of the first “R”. A typical application are grammars to build random modular forms.

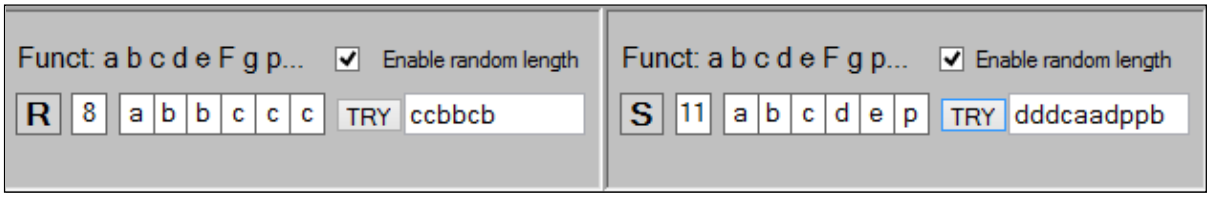


Fig. 12. Interface of “R,S,W” symbols. The number sets the maximum random length of the string, the letters the symbols that will be randomly sorted to generate the string.

2) SubSystems rules. LSystem within another LSystems. With symbols “A,B,C” the programmer can insert a complete SSRS within another SSRS. These embedded systems are called subsystems. They have independent parameters and are context sensitive, which means that they can react with their main system’ symbols. This technique enhance the parameterization, the formal complexity of the system and the level of programmability of the SSRS.

3) Parametric, programmable image or data rules. The user can load a bitmap and display the SSRS over the bitmap. An interactive palette can be used to set a rule within RGB or Brightness thresholds. It is possible to modify size, production, rotations, or bracketing.

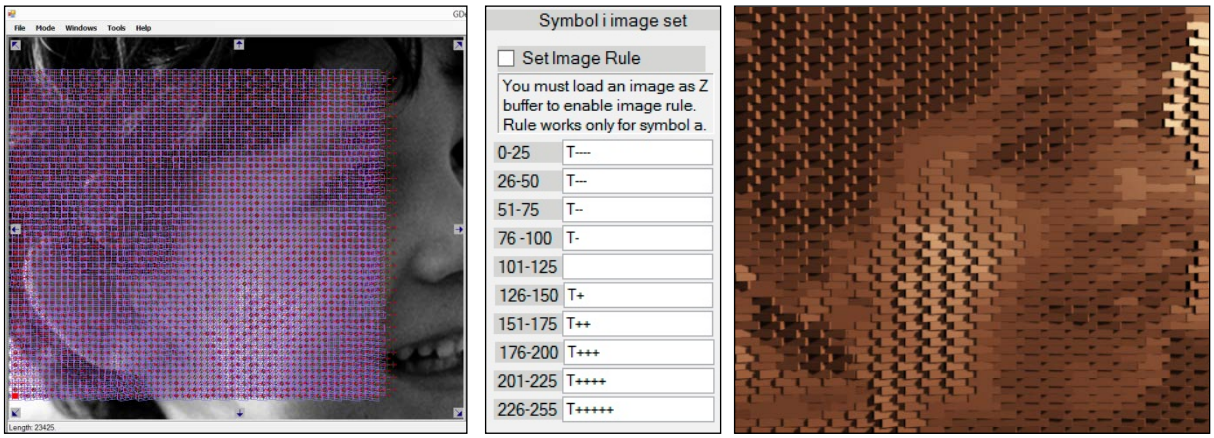


Fig. 9. Parametric brick wall using image rule with symbol “i”. The bricks are displayed over a background, and each brick will take a rotation rule that depends on the RGB values of the pixel in the same position. (“T+” means one clockwise rotation, “T---“ three counterclockwise rotations, etc.).

Flow

1) Symbol “r”. This symbol can take the rule of any symbol of any iteration step of the current grammar. It is a sort of the “go to” used in procedural programming. Usewr can set these rules with an apposite palette.

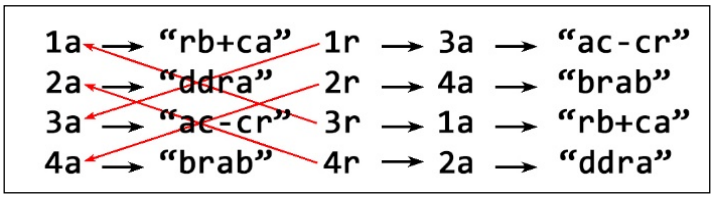


Fig. 10. Symbol “r” emulates “go to”.

2) Sequences. A sequence is a list of rules, set in any order, that user can create from scratch or assemble from a database. User can create an array of rules of any dimension, say a double array “rules[10][10]” that provides 100 different rules. The indexes numbers can be parameterized and defined by rules; this allows complex rule generation and the linking of rules between different levels of the SSRS.

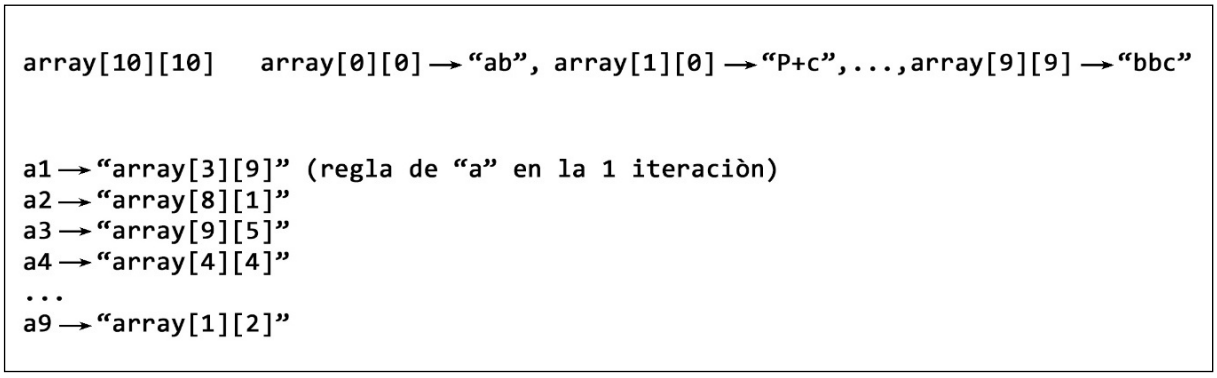


Fig. 11. Sequences. Every index’ number can be set by rules. So one symbol can decide the rule of other symbols.

3) Loops or automatic repetition of rules. It’s like the “generative” power of symbols and rules. User can set how many times the rule is applied. This allows to differentiate how symbols work. And the loop value can be set by rules and be parameterized.

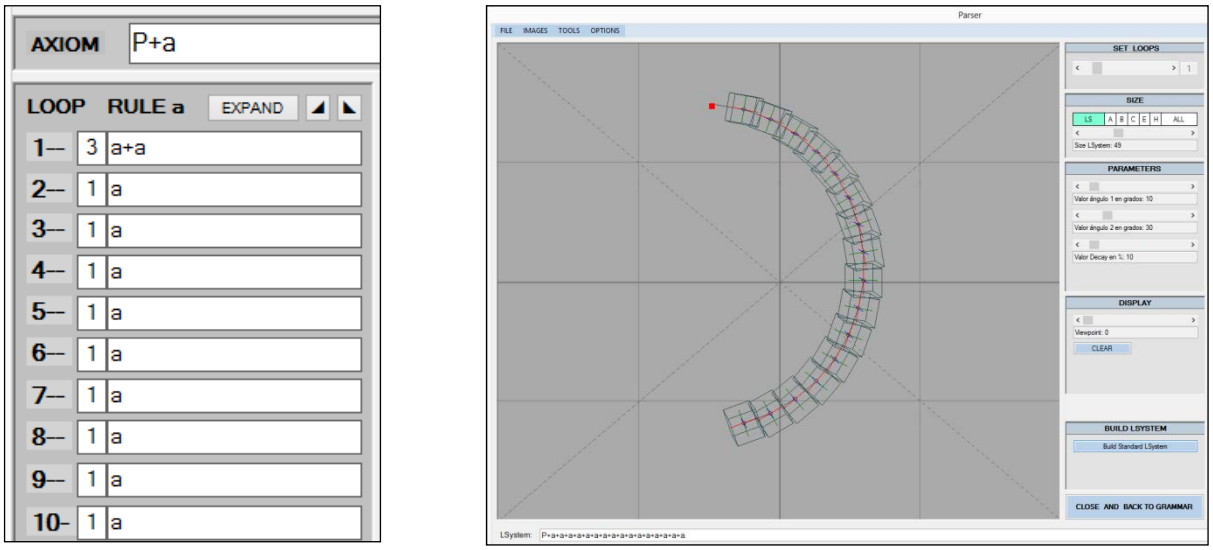


Fig. 12 - 13. The first column with black numbers indicates the substitution process step, the number in the box sets the loop, say, how many times the rule will be applied.

Interface design

The goal of the interface design is to make easy the understanding of the development of the substitution process, to facilitate the editing of rules and parameters, and to provide real time help. We used the ancient Peruvian “quipus” [14] as a visual and algorithmic

metaphor.[5]

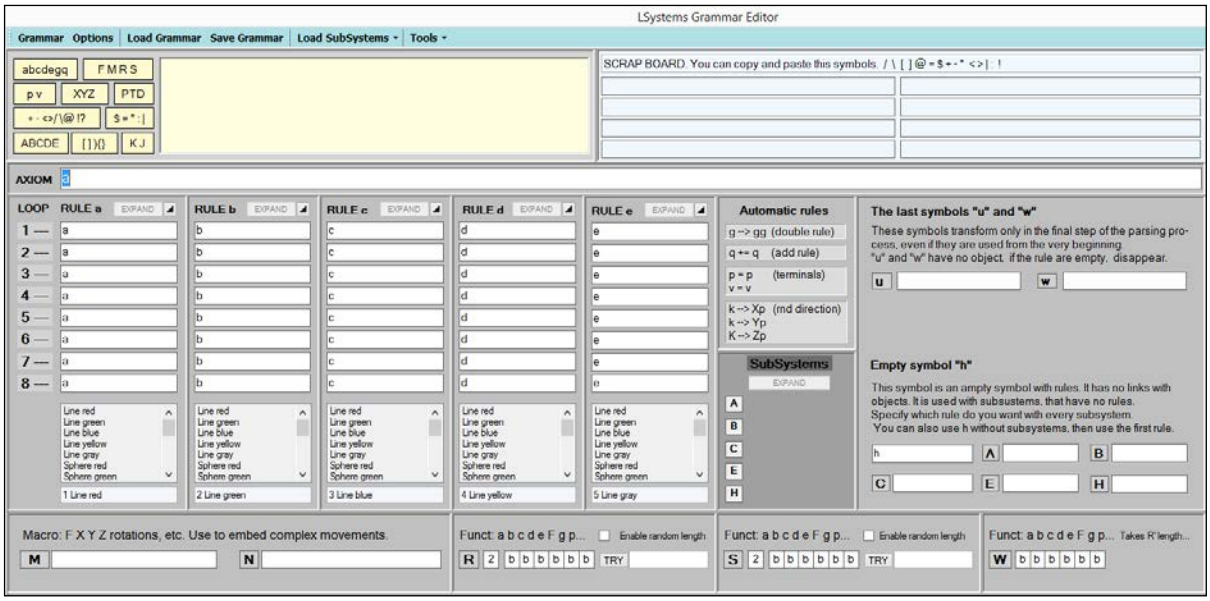


Fig. 14. The interface architecture. The idea is to make every function or task completely visible to the user. The SRSS process algorithm unfolds completely through the interface, like the pre Columbian “quipu”, an artifact made of colored stripes used to register and compute data.

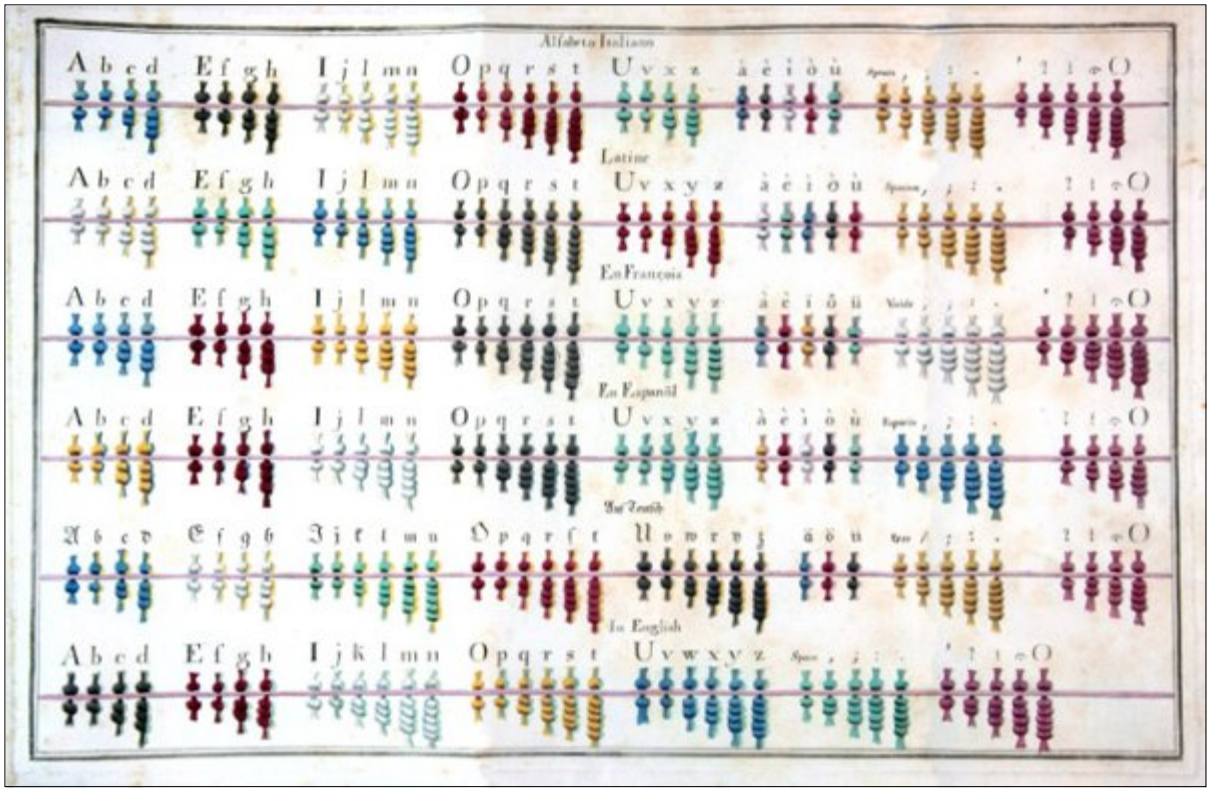


Fig. 15. The famous alphanumeric symbolic interpretation of Peruvian “quipus” by Raimondo di Sangro, prince of San Severo. 1750.

Editing

[5] Note the top down architecture of the process in the quipus’ knot pattern and the L-System’s rules of our application.

The alphanumeric string representations of 2D/3D models associated with traditional geometry information (vertex, face, solid) allows for interesting editing possibilities. The similarity with SSRS strings and DNA is obvious, and suggests hybridization options such as: chemical reactions between symbols of different SSRS, insertion of strings of symbols (like genes), removal of chunks of symbols, etc. Compression is the opposite of production: we have rules (a chunk of symbols) that we can revert to its origin. Chemical reactions between symbols can be parameterized and context sensitive.

```
string = “c+aaac+baaa[d+aaa-c]
“aaa” → “b”
new string = “cbc+bb[d+b-c]
```

Fig. 16. Editing, compress example.

```
LSystem1 = “aaaaa[P+bb]bbccc”
LSystem2 = “bbbbb[ZFFccc]aa”
LSystem3 = LSystem1 + LSystem2 = “abababababab[[P+ZbFbF]cbcbcc]caca”
```

Fig. 17. Editing, compress example.

To generate complex models, the editing solution has many advantages compared with the standard substitution process. To begin with, it’s possible to do things that otherwise are really difficult or even impossible; the genetic representation of LSystems strings, for instance, makes easy to select points of insertion and connect different forms, this operation can be done considering different data types linked to each symbol of the system.
This obviously increases the model data size, but with actual PC’s this it’s not a problem. Data structure for each symbol includes:

- Age: it gets bigger whenever a symbol maintains in the same position inside the string.

```
axiom = “ab”  ages: a=1, age b=1

a → “abb”      b → “a”          →

step 1:   “abba”     ages: a=2, b=1, b=1, a=1

step 2:   “abbaaabb” ages: a=3, b=1, b=1, a=2, a=1, a=1, b=1, b=1
```

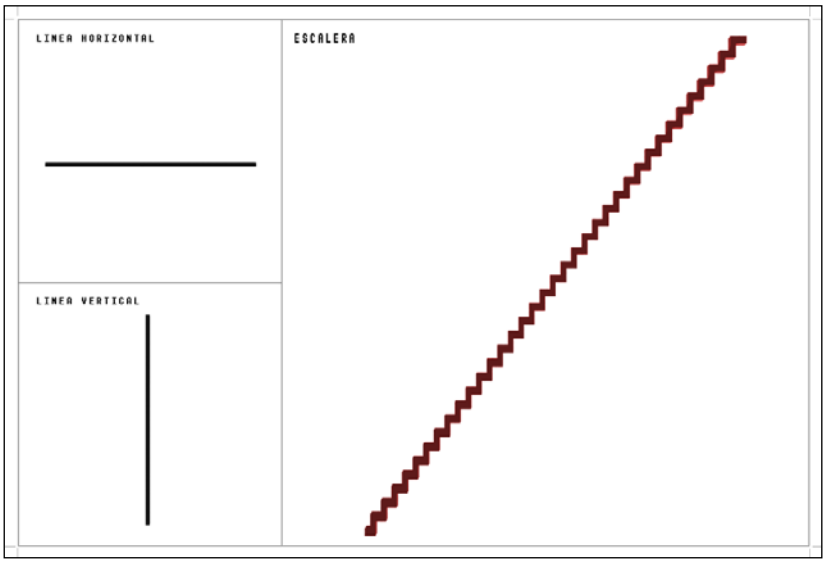
Fig. 18. Age computation.

- Distance: computes the symbolic distance from roots. It’s a powerful notation because these system are built on huge ramifications
- Density: number of objects inside a piece of string or a branch
- Homogeneity: frequency of symbols inside a piece of string or a branch

Other possible actions are:

- Symbolic geometric transformation (shrink, twist, bend, etc.). This way it is easy to insert rotations. Take the string “aaaa”, and the insertion “T+”. So the editing result will be “T+aT+aT+aT+a”, this does not depends from a substitution rule, so the effects are completely different.
- Append strings. This editing function allows to append strings at hen end or at the beginning of a branch.
- Substitute chunks. Patterns of symbols can be substituted with other patterns.

The basic case is the hybrid of horizontal and vertical lines.



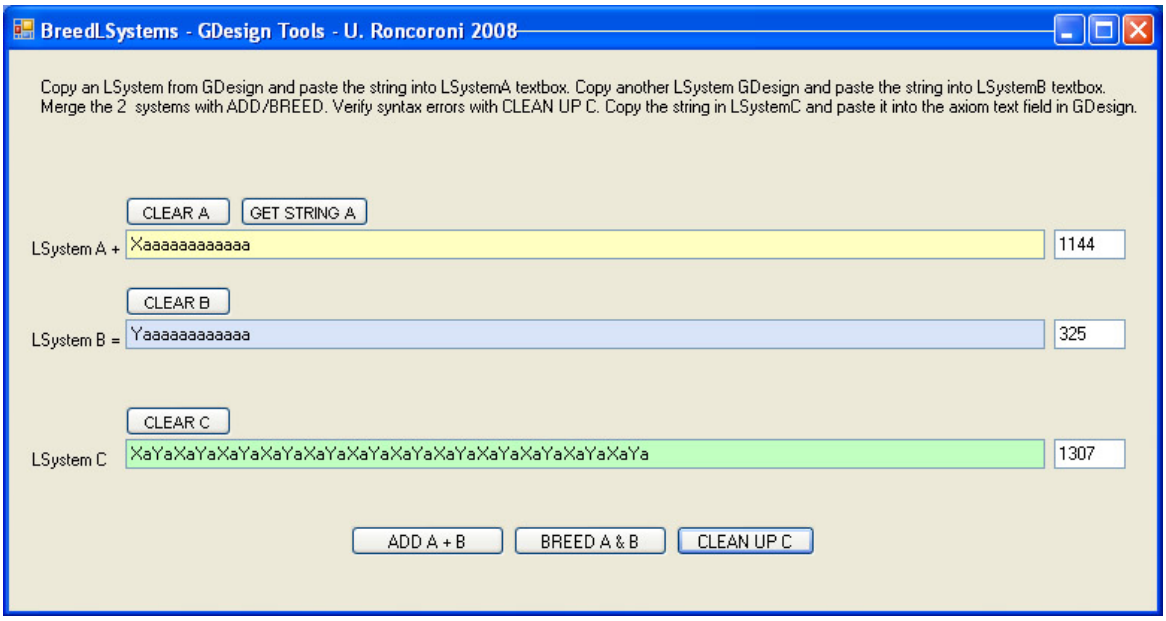


Fig. 19-20. The basics of hybrid forms: horizontal line + vertical line = stairs.

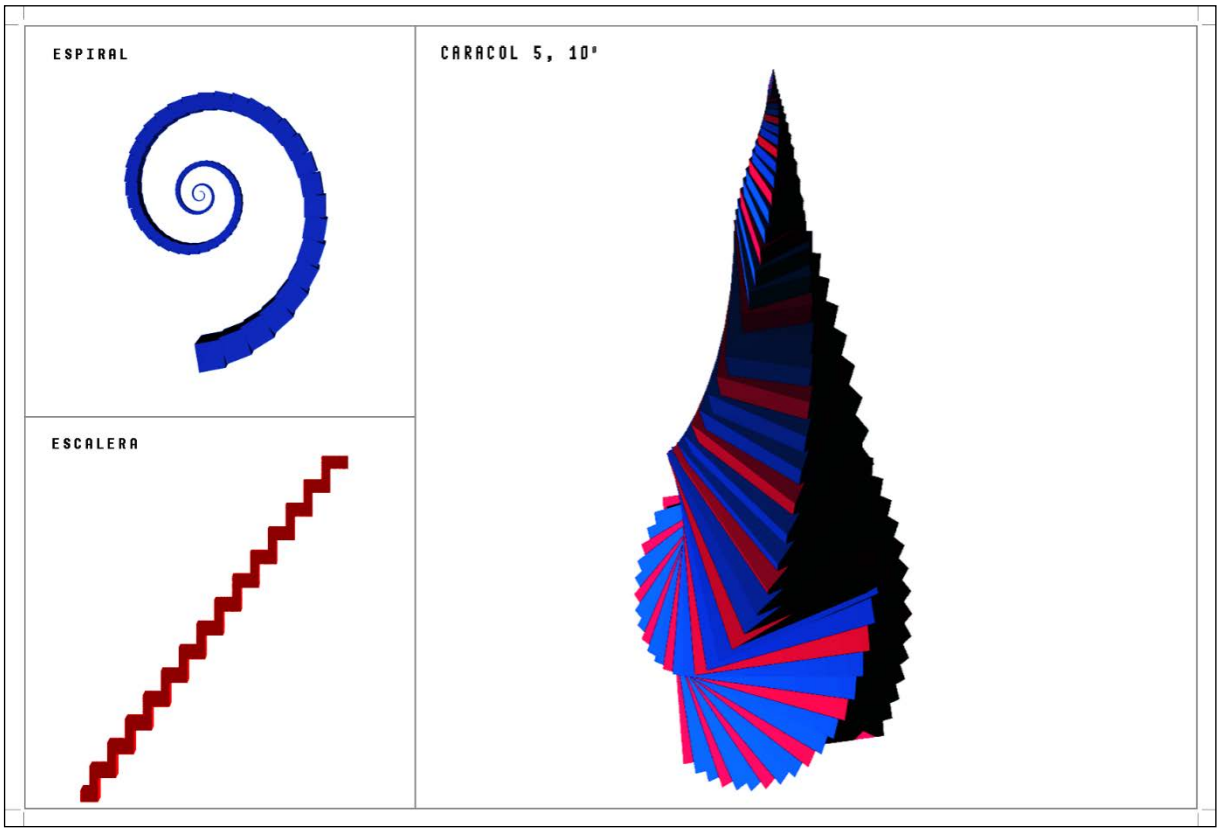


Fig. 21. A more complex hybridization example: stairs + logarithmic spiral.

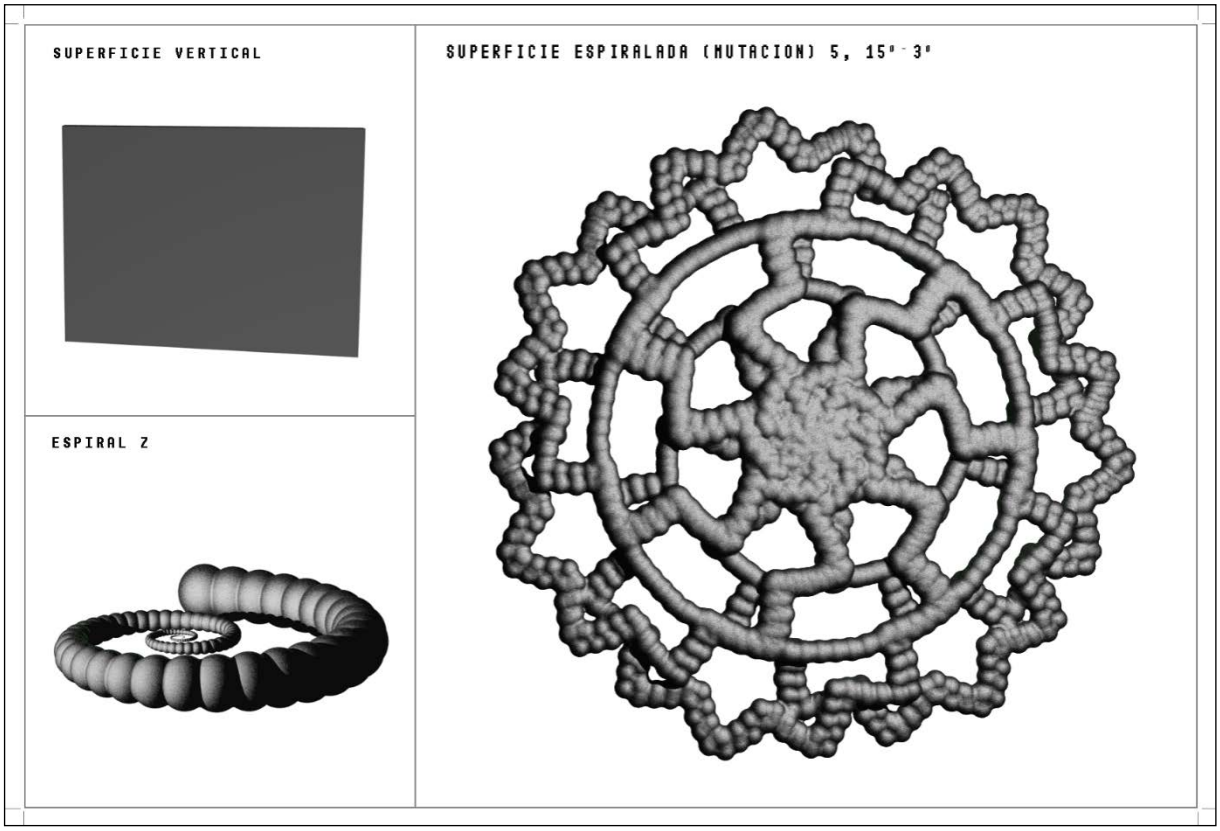


Fig. 22. The hybridization of a plane and a logarithmic spiral. Note that planes in L-Systems are made of branches of horizontal or vertical lines.

Conclusions

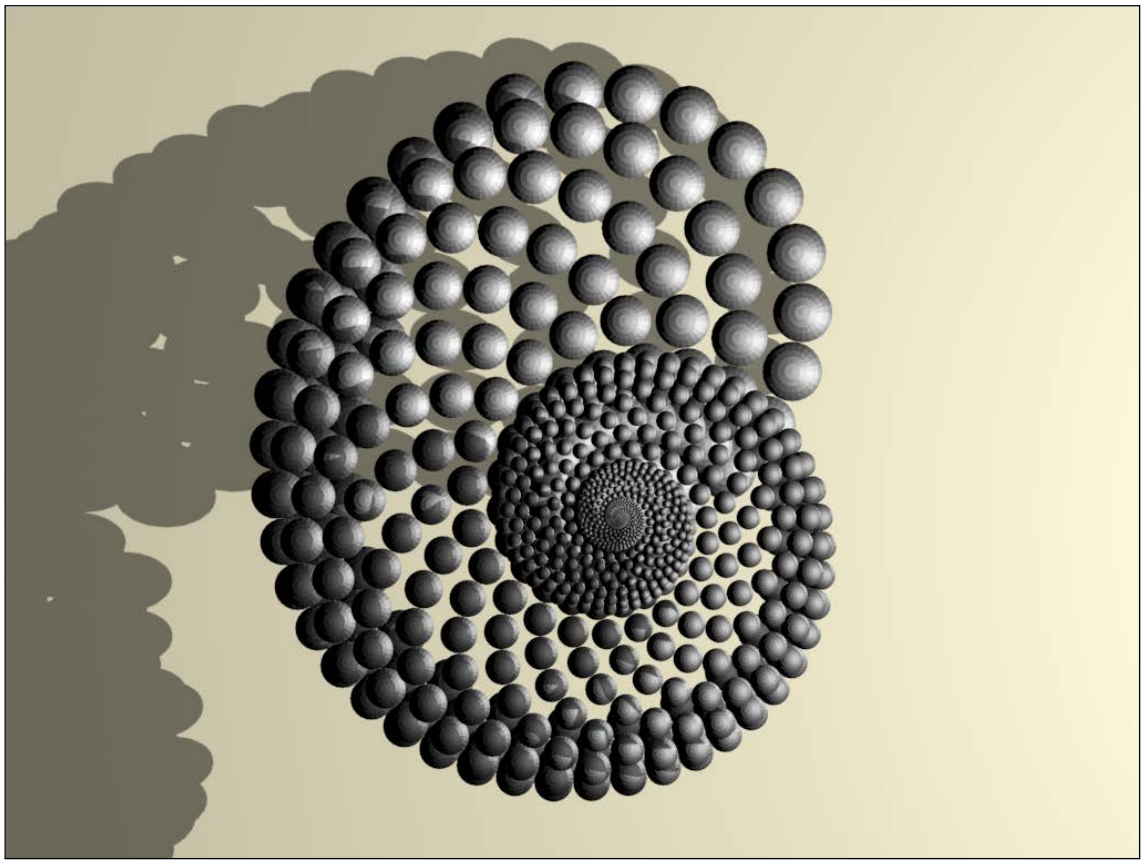


Fig. 23. 3D programmable double spiral rule with L-System.

String Rewriting Substitution Systems are powerful algorithms to generate complex forms for art, architecture, industrial and graphic design; these systems do offer a lot of advantages in terms of generative design:

- Coherence between the symbolic language of SSRS strings and the properties of form, that helps the generative development of complexity
- SRSS computational simplicity facilitates the procedural and generative understanding of form
- The symbolic notation makes easy to expand the computational power of SRSS with different metaphors, such as genetics.
- SRSS can be personalized and applied to different application and formal contexts

In this article we have tried to expand the generative capabilities of SSRS with:

- New symbols
- New rule's types
- Editing and postproduction
- Interface metaphors

Yet, the comprehension of computational procedures of recursive and substitution processes is still in its beginnings and its artistic potential are not fully understood and developed. Starting with the techniques described in this article, it should be quite easy to greatly expand the programming capabilities of SSRS and its many advantages for emergent, complex and generative designs.

References

1. Stiny, G. y Gips, J. (1972). Shape Grammars and the Generative Specification of Painting and Sculpture. En Petrocelli O. R. (Ed.). *The Best Computer Papers of 1971*. Philadelphia: Auerbach.

2. Prusinkiewicz, P. and Hanan, J. (1989). *Lindenmayer Systems, Fractal, and Plants.* New York: Springer-Verlag.

3. Prusinkiewicz, P. and Runions, A. *Computational models of plant development and form. New Phytologist* 193, pp. 549-569, 2012.

4. Reynoso, C. (2008). *Diseño artístico y arquitectónico con gramáticas complejas*. [en línea]. http://carlosreynoso.com.ar.

5. Fišer, Marek. (2012). *L-Systems on line*. Bachelor Thesis. Faculty of Mathematics and Physics. Prague: Charles University.

6. Di Napoli, Giuseppe. (2011). *I principi della forma*. *Natura, percezione e arte*. Torino: Einaudi.

7. Flake, Gary W. (1998). *The computational beauty of nature*. Cambridge: The MIT Press.

8. Stevens, P. (1986). *Patrones y pautas en la naturaleza*. Barcelona: Salvat.

9. Ball, P. (2001). *The self-made tapestry. Pattern formation in nature*. New York: Oxford University Press.

10. Crousse, V. (2011). *Reencontrado la espacialidad en el arte público del Perú*. Tesis doctoral en Espacio público y regeneración urbana: arte, teoría y conservación del patrimonio. Barcelona: Universidad de Barcelona, Facultad de Bellas Artes. [en línea]. http://www.tdx.cat/handle/10803/1551.

11. Ruiz-Montiel, M., Boned, J., Gavilanes, J., (...), Mandow, L., Pérez-de-la-Cruz, J.-L. (2014). *Architectural Design with Simple Shape Grammars and Learning.* Inteligencia Artificial 17 (54), pp. 21-29.

12. Jesus, D., Coelho, A., Sousa, A.A. (2016). *Layered Shape Grammars for Procedural Modelling of Buildings*. Visual Computer 32 (6-8), pp. 933-943.

13. Leyton, M. (2001). *A Generative Theory of Shape.* Berlin: Springer-Verlag.

14. Di Sangro, R. (1750). *Lettera Apologetica dell'Esercitato Accademico della Crusca contenente la Difesa del libro intitolato Lettere d'una Peruana per rispetto alla supposizione de' Quipu scritta alla Duchessa di S**** e dalla medesima fatta pubblicare.* Napoles, s/e.

SHAPE GRAMMARS PROGRAMMING BASED ON NATURAL FORMS AND PREHISPANIC ARTIFACTS

Umberto Roncoroni - Universidad de Lima. hroncoro@ulima.edu.pe
Veronica Crousse - Pontificia Universidad Católica del Perú. vcrouss@pucp.edu.pe

Step 1- NATURAL FORMS

Morphologic research of natural forms photographing landscapes, plants, rocks and microscopic specimens.

Step 2- PREHISPANIC ARTIFACTS

Morphologic research through photographs of prehispanic landscape modelling, constructions and artifacts such as textiles, ceramics, metal objects or mural engravings.

All these photographs were taken in several trips in coastal, Andean and Amazonian territories of Peru between 2008 and 2015.

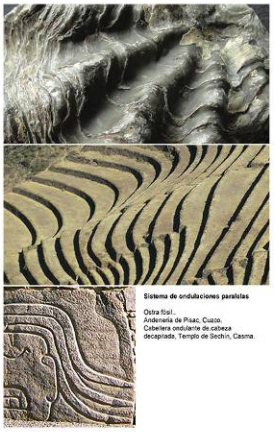
Goal: Record of natural and prehispanic artifact's morphogenesis to gather structural patterns.

Step 3 - COMPARATIVE RECORDS OF NATURAL FORMS AND PREHISPANIC ARTIFACTS

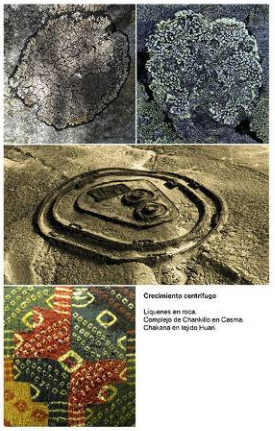
Analysis of the formal relationships between structures founded in nature, prehispanic architectures and objects.

Goal: Proving structural correspondence andsimilarities between repetitive and recursiveprocesses of nature and prehispanic manufacturing.

Estructuras espirales
Huella digital.
Brote de hoja de helecho.
Acueducto de Cantayoc.
Tejido precolombino.



Sistema de ondulaciones paralelas
Ostra floal.
Andenería de Pisac, Cusco.
Cabellera ondulante de cabeza decapitada, Templo de Sechín, Casma.

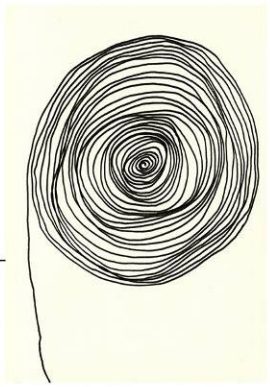


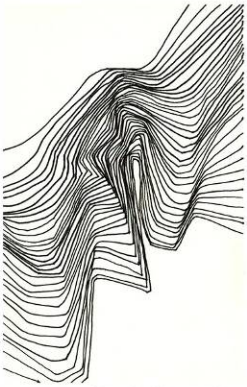
Crecimiento centrífugo
Líquenes en roca.
Complejo de Chankillo en Casma.
Chakana en tejido Huari.

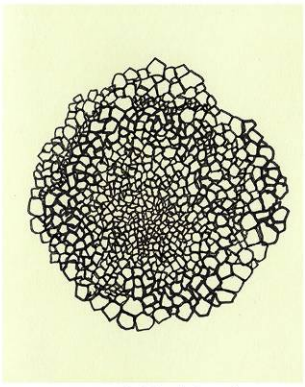
Step 4 - ANALYSIS OF FORMS THROUGH ALGORITHMIC HAND DRAWINGS

Procedural hand drawings as an exploratory process to discover the shape grammar rules behind forms and patterns found in previous steps.

Goal: Understanding of morphogenicprocesses and rule's deduction to develop new algorithms for Shape Grammars and L-Systems.







Step 5 - WRITING THE ALGORITHMS, DESIGNING SOFTWARE

The procedural rules discovered through the hand drawing process are used to design the software's algorithms.

Algorithms derive from the breeding of natural forms, prehispanic artifacts and hand drawings.

Goal: New tools for Shape Grammars and L-Systems programming.

Estructuras espirales
Dibujo algorítmico a mano alzada.
Proceso:
Parte una línea del borde del papel y a un cierto punto se empieza a plegar estableciendo un recorrido centrípeto que la lleva a enrollarse en sí misma. Las líneas intentan estar lo menos distantes entre ellas, pero el dibujo a mano alzada crea irregularidades que se mantienen o acentúan con la repetición de las líneas, configurando una espiral no perfectamente circular.

Sistema de ondulaciones paralelas
Dibujo algorítmico a mano alzada.
Proceso:
Se inicia en la parte alta del papel trazando una línea con una ligera ondulación. Se continúa hacia abajo, dibujando líneas que intentan adaptarse a su entorno inmediato y ser paralelas a su vecina inmediata. Esto quiere decir que las irregularidades y variaciones de cada línea, introducidas por la casualidad y la impresión que aporta el dibujo a mano alzada, va creando ondulaciones totalmente distintas a las de la línea inicial.
Es un dibujo emergente, que mediante la repetición y sucesión c líneas, va creando un sistema complejo de ondulaciones.

Crecimiento circular centrífugo
Dibujo algorítmico a mano alzada.
Proceso:
Se inicia en el punto central del papel dibujando una forma poligonal de 4, 5 o 6 lados. La arista donde se cierra el primer polígono es el punto de partida del siguiente, que cuesta ubicado inmediatamente adyacente al primero y sin dejar espacio.
Este proceso se repite una y otra vez, siendo la circunferencia ocupar el espacio del centro hacia la periferia sin que el trazo se disjunque del papel.