Generative Techniques for Building Virtual Objects

Grigore Albeanu

Department of Mathematics and Informatics
Spiru Haret University
13, Ion Ghica Str., 030045, Bucharest, ROMANIA
E-mail: g.albeanu.mi@spiruharet.ro

Abstract
Many generative methods in arts and science are available. This paper presents a review of various generative methods oriented to the building of virtual objects useful in virtual reality applications. Grammars, automata, and evolutionary strategies are some classes of generative methods. The paper is dedicated both to Alan Turing and John von Neumann.

Keywords: generative grammars, shape grammars, cellular automata, L-systems, evolutionary computing, DNA computing, membrane computing, GML

1. Introduction
Building virtual objects is an important task in virtual environment development. Nowadays, for a large variety of fields, including education, science, and entertainment, the researchers, designers, and users are interested in finding ways (approaches) to design virtual entities. One approach is based on the generative way of thinking which is rules-based. Also, Eckert (1999) considered that, “the term generative systems can cover a wide range of different automatic design tools”. The mentioned author has included “the genetic algorithms, shape grammars and other evolutionary methods share the characteristics that they create a large number of designs, which are pruned using a fitness function.” Our vision on generative systems includes also generative grammars, cellular automata, DNA computing, membrane computing, and other nature-inspired computing mechanisms.

As McCormack et al. (2004) remarked, the modern world asks for a new way of object design adapted to new electronic systems and devices available today. Viewed as a dynamic process, the generative methodology produces new forms in a sequential or parallel process using production rules. Alternatively, we say that a computation mechanism is used. The scientific literature refers to many computation mechanisms like Turing machines, Markov algorithms, Kleene recursive functions, Post production systems, all of them converging on the same class of computational capabilities, those of effective computation of Church. According to Eberbach et al. (2004), the Turing’s thesis claims that “whenever there is an effective method for obtaining the values of a mathematical function, the function can be computed by a Turing machine”. This paper did not detail on Turing machines. However, the reader should be advised on the structure of the automatic machines proposed by Turing: a one-dimensional erasable tape of infinite length, capable of storing symbols (one per cell); a read/write tape head moving to the left or to the right depending on the program, a control mechanism (working in a bounded number of states), and a transition table (the program), specifying the next action and the new state of the machine given an initial state and a symbol under the tape head. More details in computability and its complexity are given by Cooper (2003). An excellent material about the Turing’s ideas and models of computation is presented by Eberbach et al. (2004).
Choosing a particular computational mechanism depends on the field under study: formal grammars (of Chomsky - 1965) are used to generate configurations by production rules, finite state machines use transitions between system states (which can be thought also as configurations) but more relevant for technical or nature-inspired systems. However, a generative mechanism should have some key properties like: (1) ability to generate a large number of objects (configurations) including novel structures, behaviours, final objects or relationships; (2) ability to generate interactions object-environment; (3) self-maintenance and self-repair capability. Such mechanisms are called intelligent/smart generative systems.

This paper is a review on most important generative ways of thinking providing high impact on virtual objects design. Firstly, the philosophy of generative thinking is presented. Next, the methodologies for generative design are explained and illustrated by adequate generative mechanisms. Finally, comments and concluding remarks will be given.

The next section discusses generative grammars and their capability to build virtual objects.

2. Generative grammars

Basically, grammar based generative mechanisms are specified by an alphabet (characters, shapes, etc.), an initial configuration (in theory the start symbol, or the axiom, a string of characters etc.), a set of production rules used in a string-rewriting process to generate configurations (strings of characters, shapes, etc.).

Kirsch & Kirsch (1986) illustrates the usage of formal grammars in design, in fields like architecture and graphic design. Borrowing from structural linguistic terminology, they consider both a deep structure and a surface structure associated to every artwork. The surface structure refers to observable properties like: texture, variation of media, line attributes, colours and their relationships. In general these properties are related to aspect and topological attributes. When refer to the deep structure, Kirsch & Kirsch mind structural aspects related to overall composition and the work organization in two or three dimensions. The recursion property (an unpleasant one for lexical/compiler generators) was found very useful in art design. It is known that recursion appears when a formula production leads to direct or indirect its invocation during structural generation. The mentioned authors described a formal grammar generating Richard Diebenkorn’s Ocean Park paintings.

Another extension of phrase structure grammars for specification of paintings and sculptures is represented by shape grammars introduced by Stiny & Gips (1971). Therefore, a class of paintings was defined by a specification S of a class of shapes consisting of a shape grammar defining a language of two-dimensional shapes and a selection rule, and a specification M of material representations for the shapes defined by S and consists of a finite set of painting rules and a canvas shape. Shape grammars are defined over an alphabet of shapes and can generate n-dimensional shapes. An initial shape is given. Recursively, the shape rules are applied to the shape rules, in order to obtain new shapes.

The obtained shape is generated by applying a rule as follows: initially is necessary to find a part of the shape that is similar (geometrically speaking) to the left side of some rule (production); then a geometric transformations (scaling, rotating, translating, shearing, etc.) is found to be applied for the identified part which is similar to the left side of the rule, and finally, the selected transformations will be applied to the right side of the rule for the corresponding part of the given shape.

There are terminal shape elements (impossible to be erased after a previous addition to a shape), and markers (non-terminals). For shape grammars the rules are shape specific and rule replacement only focuses on the partial transformation of a shape.

Boundary solid grammars (BSG) were introduced by Heisserman in his PhD thesis, and presented later in the journal IEEE Computer Graphics and Applications, March 1994. The BSG approach consists of an initial solid and a set of rules generating multiple configurations.
Solids are built according to the graph-based boundary representation. Solid rules derive a labelled boundary graph from another. The boundary graph is a topology graph (with nodes like vertex, vertex-use, edge-half, edge, loop, face, shell, shell-use, solid, model, and arcs representing the adjacency of nodes) with coordinate geometry (given by a function) associated with each node. Labels are attached to nodes in order to associate non-geometric data useful not only for searching, but also for various computations (solid’s mass as an example) and texturing.

The BSG formalism was implemented by the Genesis system consisting of a custom boundary-representation solid modeller, a database for non-geometric information, solid visualization methods, and a graphical user interface.

Cui & Tang (2012) introduced the Dynamic Shape Representation (DSR) as a new representation of 3D shapes. A basic rule using fundamental primitives is called Elemental Rule (ER). Moreover, an ER can consists of a finite set of shape rules of the form (u, v), where u and v are the shapes being included in the permitted primitive shape set with an orientation, called ER(i). The DSR model uses a DSR initial shape, DSR shapes, and a DSR shape grammar. A DSR initial shape is a collection of definite primitive shapes by union operation, while a DSR shape is a finite set of shapes being generated by using a family of ERs in a specific order: ER(0), ER(1), ..., ER(i). A DSR shape grammar has four components (S, L, R, I): the finite set S of DSR shapes, the finite set L of symbols, a set of rules denoted by R, and an initial DSR shape, denoted by I. The DSR shapes can be generated using CSG (constructive solid geometry) grammars for Boolean expressions of geometric primitives and Boolean operations. Boundary solid grammars can also be used to generate DSR shapes.

Finally, in the end of this section we remind L-systems (see Prusinkiewicz and Lindenmayer(1996)) as parallel generative grammars useful to model plant development, branching tree structures, and other fractal artefacts. Culik & Kari (1993) described a “turtle” interpretation of L-systems, with the turtle drawing squares.

3. Cellular automata for pattern generation

Cellular automata are discrete dynamical systems used for computer simulations of various natural phenomena, as Culik & Kari (1993) wrote. However, the first developments of cellular automata were realized by John von Neumann and Stanislaw Ulam at Loss Alamos National Laboratory, in 1940.

For the subject under development, two-dimensional cellular automata will be described. The infinite Euclidean plane is divided into unit squares (cells) indexed by using integer coordinates. Each cell, at any moment of time, is in one of states belonging to a finite set S. At discrete time steps the cells alter their states synchronously following some local transition rules. A local transition rule specifies the new state given the old state and the neighbours of the cell. Two types of neighbourhoods are used: the Moore neighbourhood (the cell and its eight neighbours), and von Neumann neighbourhood (the cell and its four neighbours). The automata evolve from one configuration (the state infinite matrix of cells) to another. A colouring function can be used to give the colour for each state.

Another approach to generate patterns (tessellations) uses finite automata and iterative matrix substitution. It is well-known that the accepted language of finite automata can be described by regular expressions and generated by regular phrase grammars. A matrix substitution system (MSS) consists of an alphabet (finite set), a set of productions, an initial symbol, and markers to represent black squares in the substitution matrices. An MSS can be made more convenient by allowing rotation or flipping of the substituted submatrices.
4. Bio-computing inspired models
DNA recombination operation can inspire designers of virtual objects. The splicing operation (see Daley & Kari (2002)) can be used to build a generative mechanism also. Given a set of axioms (strings, basic shapes) and a set of spicing rules, a generative sequence can be obtained starting from axioms and the iteratively use of splicing rules on already built elements.

P-systems, introduced by Paun (2003), can be used in generative art and procedural modelling of plants. Wagy, McDermott & Keefe (2011) describe two preliminary investigations into the utility of P-systems for graphics. They also used the turtle geometry approach.

Virtual artefacts can be generated by various evolutionary computation approaches. The designer should find a good representation of individuals (strings, or matrices), design variation operators (mutation and/or recombination), and define a selection/reproduction mechanism. Cellular automata can evolve using genetic operators. Also, genetic operators can be used to evolve L-systems. The term grammatical evolution is used. Selection, breeding, and mutation are used by Beaumont & Stepney (2009) to evolve simple L-systems by grammatical evolution.

5. The Generative Modelling Language
Before concluding, let us have a look on GML (the Generative Modelling Language) which combine simple shape construction operations as Havemann (2005) shown. It is a “shape programming language”, providing a number of operators for creating 3D models (polygons, b-reps, subdivision surfaces).

As a stack-based (postfix) language, GML is a powerful language for the representation of procedural models, the design of virtual objects becoming focused on rules for transforming primitive objects instead of focusing on combining these primitives. The OpenGL-based runtime engine is used by GML to work like a viewer with an integrated modeller, in order to overcome the usual separation of 3D modelling from interactive visualization.

6. Conclusions
The generative modelling was investigated above. Not only classical generative mechanisms like Chomsky phrase grammars were considered, but also more suitable for virtual reality applications shape grammars, boundary solid grammars, cellular automata, DNA inspired operators and membrane computing were identified as interesting approaches in virtual assets modelling, including music.

Finally, the GML was identified as a “shape programming language” useful to create virtual worlds by procedural modelling which is a better alternative to the usage of VRML based on declarative modelling.

7. References


