AGENCY GP: Agent-Based Genetic Programming for Spatial Exploration

PETER TESTA
Department of Architecture, Massachusetts Institute of Technology

UNA-MAY O'REILLY
Artificial Intelligence Laboratory, Massachusetts Institute of Technology

SIMON GREENWOLD
Emergent Design Group, Massachusetts Institute of Technology

Abstract

AGENCY GP is a prototype for a system using genetic programming [GP] and software agents for architectural design exploration. Its software structure is noteworthy for its integration into a high-end three dimensional modeling environment, its allowance for direct user interruption of evolution and reintegration of modified individuals, and its agent-based evaluation of fitness.

Background and Motivation

AGENCY GP is the software arm of a larger project called AGENCY currently under way at MIT's Emergent Design Group (web.mit.edu/arch/edg/about.htm). The project seeks architectural responses to the radical transformation that business organizations are currently undergoing. The pace of organizational change is being driven by the rapid development of commercial technology, global markets and reengineered institutions. This constant need to change gives rise to organizational structures that are no longer stable, but continuously adapting to their shifting environments. Such structures can be said to be "emergent" and describe many of today's commercial and governmental organizations.

Vertically structured office buildings no longer provide the model for most businesses. With the advent of widespread use of telecommunications, information technology, and corporate reorganization in the 1990's, new forces are actively reshaping the architecture of office buildings. There is a shift in the United States toward research and development, management and finance, consultancy, and the culture industry, productive activities less prone to standardization and bureaucratization. Driven by the demand to improve office productivity, businesses and organizations have begun to experiment with a variety of alternative officing methods. However, there exist no working models of an intelligent adaptive architecture. The AGENCY project focuses on application-oriented basic research to develop new design software that generatively models the complex interactions of physical space and information technology within emergent organizations. Using the research software this project will generatively design and test spatial systems and work environments.

We have chosen genetic programming to address these challenges because the strengths of the model are well matched to our system's desired characteristics. First, we are aware of the impossibility of modeling emergent organizations deterministically. To try to design a deterministic algorithm for the creation of workspaces would certainly fall victim to our inability to name every constraint the problem entails. Therefore we look to the stochastic, self-organizing solutions Aritificial Life, and in particular GP affords, to construct solutions that are consistently sensitive to complicated interactions that a user need not explicitly codify. Second, we are interested in the genetic model's ability to offer a user an entire population of solutions to peruse and potentially to revaluate. The process we are involved in is not a simple optimization with a single goal, but has many potential fruitful avenues of exploration. The multi-tracked exploratory process of population evolution provides a designer multiple alternatives with which to interact at any point. Our goal is to develop GP as a design partner, offering options that would otherwise not come to light.

AGENCY GP integrates three technologies with entirely separate pedigrees—genetic programming, software agents, and Alias|Wavefront's Maya—to form a single system for design evolution.

Genetic Programming

Genetic programming is a methodology for machine learning that grew out of the field of genetic algorithms. The genetic algorithm [GA] model takes as its organizing metaphor, the biological processes of evolution. The system begins by creating a population of “individuals” each represented by a “genotype.” In the case of GA, the genotype typically consists of a fixed-length string of symbols, which can be decoded and expanded into a phenotypic individual in the same way that a biological genotype is phenotypically expressed in an organism. These individuals then undergo a simulated process of evolution (Koza).

The natural process of sexual reproduction allows for the recombination of genes from one generation to the next by the “crossing-over” of genetic material. This allows for dramatic differentiation of offspring from parents. An analogous method of crossover occurs within the GA system’s population as pairs randomly mix their genotypes to breed a subsequent generation of individuals. Mutation of an individual’s genotype is also a common occurrence in reproduction, and it occurs randomly.

However, as in natural selection, not all individuals have an equal probability of transmitting their genes to the next generation. A population of individuals’ phenotypes are ranked by their “fitness,” and the associated individuals are selected to breed in rough proportion to their fitness ranking. Fitness can be calculated by any consistent measure. In the case of AGENCY GP, fitness is derived from the combined efforts of multiple software agents.

The control loop for a GA system begins by creating an initial population of randomly generated genotypes. Then the genotypes are interpreted to produce phenotypes, which are ranked by fitness. A new population is created by the breeding and mutation of individuals, and the old population is discarded. Although all of the genetic operations that produce one generation from the previous are stochastic, the pressure of selection is assertive, and the result of multiple generations is a population that is on average, far fitter than its predecessors. GA is of wide application to problems of multivariate optimization.

Genetic programming [GP] is an extension of the genetic model, in which the genotype is expressed as a variable-length list or tree of commands in a simple language. The phenotype then represents the execution of those commands. When genomes in GP are mutated and crossed over, commands are inserted or deleted in the genotype or their parameters are changed. Much of the success or failure of an application of GP depends on the design of the internal language that constitutes the genotype. In the case of AGENCY GP, we have developed a language that operates on primitive shapes and combines them with others to build up more complicated designs that constitute the individuals in a population.

It is worth noting that typically in a GP system, it is difficult or impossible for a user to halt evolution, directly modify an individual, and resume evolution—a procedure known as “interruption, intervention, and resumption” [IIR]. This is because in languages which contain conditional structures, it is not possible to map changes backwards from phenotype to genotype. The cascade of compounding consequences that determines the course of the genetic program destroys the potential for a one-to-one mapping of genotypic to phenotypic feature. The language we have developed for AGENCY GP, however, is non-branching, simple enough that it will allow for IIR.

Advantages of GP

We chose GP to be a foundation of Agency’s software model because several of its characteristics fit well with the exploratory role we hope Agency will fill. That GP offers a population of designs means that a designer using the system is always presented with multiple alternatives from which to proceed. There is not a single iteratively improved candidate design.

The GP model also provides a constant process with which to interact rather than a simple product. In fact, the evolutionary process does not have a specific end product, nor will a GP effectively find a globally “optimal” solution to a problem with as many variables and constraints as ours. The process will, however, continue to explore new designs as long as a user wants to watch it run. It will be possible for a user to exert control at several levels, from direct manipulation of an individuals’ phenotypes, to nudging the course of evolution by changing its parameters.

That the genetic regeneration process is fundamentally non-purposive and stochastic means that designs will be generated and evaluated that do not fit a designer’s preliminary notions of good fitness.

Should such designs consistently find favor with evaluating agents, the user will be forced to reevaluate assumptions about what makes a design worth considering.

Agents

The second software technology we integrate into AGENCY GP is the software agent. Agents are defined differently in different sources, but general definition we are using is "a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on behalf of its user" (Nwana, 1996). Agents-based models are finding application in fields that require distributed attention and interaction, such as electronic commerce, system administration, network management, and information retrieval (Milojicic, 1999). Agents are autonomous processes that can identify themselves, run concurrently, and interact with each other and their environment. Agents can be supportive of one another, neutral, or antagonistic. There may be multiple instances of the same agent scattered widely—this might be a good way intelligently to collect data from a massive unindexed source like the Internet—or each agent may have an entirely different instruction set.

We chose to implement our fitness evaluations by means of distributed software agents inhabiting the candidate designs. Once the designs have been interpreted from genotype to phenotype, multiple agents are instantiated to test the designs for various criteria. Some agents may represent actual users of the space, while others will be interested in issues such as fire-code compliance, or energy efficiency. Virtually any criterion for evaluation can be coded and dropped in as an agent to our framework. We can specify that workspaces require a certain quotient of natural light or that circulation spaces desire width enough to allow for conversation. Using agent-based evaluation, we will able to model management structures and determine their influence on potential designs. An agent may represent the pattern of a group, its needs for privacy, meeting space and collaborative surfaces, or it may undertake the concern of management structure or productivity. Agents will individually and collaboratively rate the design, and their feedback will be incorporated into a measure of overall fitness.

Advantages of Agents

What agent-based fitness allows for is a modular structure for the integration of multiple criteria for fitness. We have abstracted the agent structure so that new agents may be developed and employed for new applications without rewriting the entire AGENCY GP system. In fact, we are free to write agents that have non-spatial concerns. For instance, we may implement an agent whose interest is organizing teams of workers. The spatial effects of such an abstracted agent cannot be determined in any way other than to run the system under this agent's selective pressure. Since the descriptive language and representation of the individual designs is spatial, the system effectively translates non-spatial constraints into spatial hypotheses. This agent-based evaluation of fitness is well suited to expressing the conflicting, non-linear, multi-level spatial requirements of emergent organizational structures.

Maya

We are implementing our software as a plug-in extension of Alias|Wavefront's Maya package, a leading tool for the creation and visualization of complex virtual environments. Maya is of widespread use in the fields of three-dimensional animation for film and broadcast, but has only recently begun to see broad application to explorations of morphology such as AGENCY GP. Maya's open architecture allows for software developed using all modern high-level languages features to exploit the power of its inbuilt library of three-dimensional operations. Any three-dimensional form is expressible in terms of Maya's available spatial transformations and Boolean operations. The Maya platform allows us to abstract the representation of three-dimensional forms so that we can operate freely on them without concern for the complexity of the underlying geometry. The internal language that we have developed for evolution in the GP system manipulates spatial constructs at the same level that a designer does when working in a three-dimensional modeling system. Therefore the individual commands of the language are meaningful to and useful by designers, ensuring the efficacy of IIR.

Language and Representation

Our implementations of the GP's main generational loop, selection, mutation, and crossover are conventional. The individuals in the population are fairly complicated structures, each containing enough information to describe a complete design. We use a combination of Maya and C++ objects for the internal representation. A user begins the process with a Maya scene with one or more closed NURBS (non-uniform rational b-spline) curves selected. These curves should be coplanar but may be of any closed form and may intersect. This initial set of curves acts as the seed for all subsequent evolution.

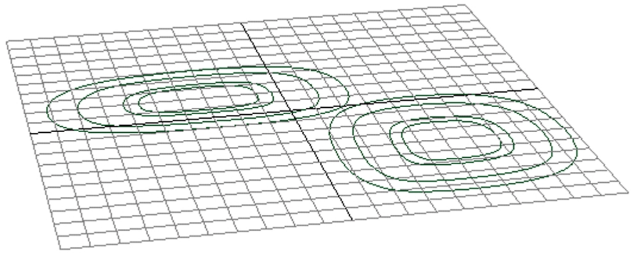

Fig. 1. NURBS Curves in Maya

In the interpreted phenotypic representation, such curves will be extruded into space.

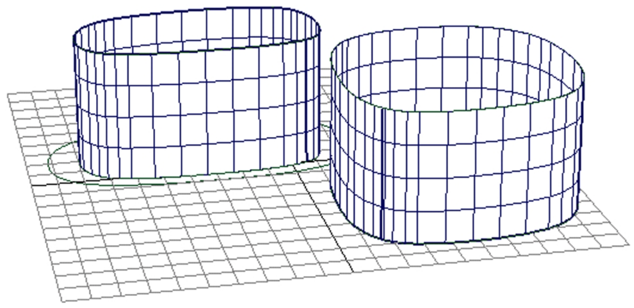

Fig. 2. NURBS Curves Extruded Into Space

When the AGENCY GP is invoked from within Maya, a C++ object is permanently attached to each NURBS curve in a design. This C++ construct contains evolvable values pertaining to the shape and architectural function of the region the curve encloses. The object also contains an evolvable sequence of operations in our GP language to be applied to this curve. Each curve is given a height of extrusion and treated as a NURBS surface extending from an evolvable starting height into space.

The operations in our language are simple but powerful transformations of these NURBS surfaces. The images below demonstrate the operations of our language applied to the lefthand NURBS surface from the scene in figure 2.

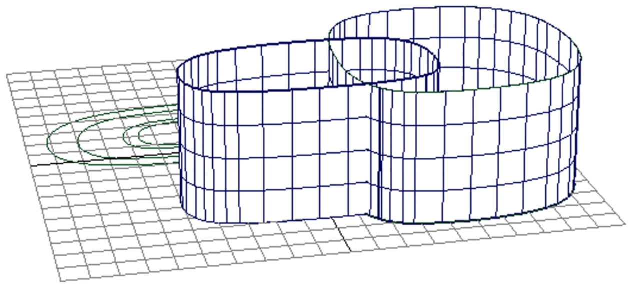

Fig. 3. TRANSLATE (X, Y)

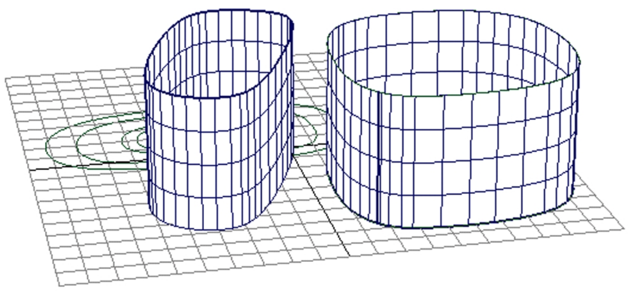

Fig. 4. ROTATE (DEGREES)

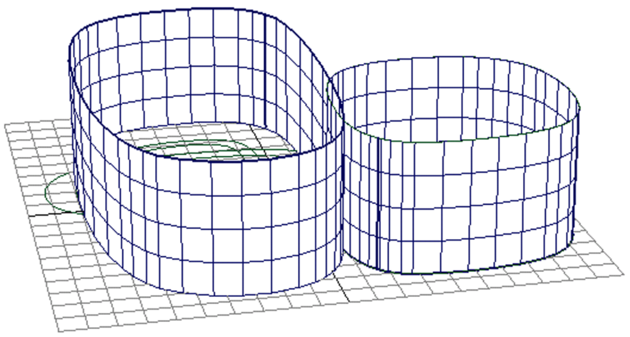

Fig. 5. SCALE (X, Y)

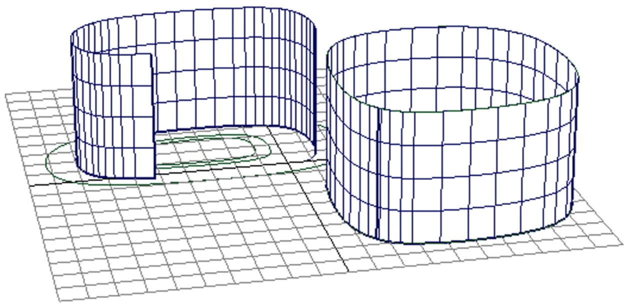

Fig. 6. CUT (START, STOP)

These operations form the core commands of our language. Mutation may consist of addition or deletion of an operation, or the change of a parameter. Execution is strictly linear; there is no facility for conditionals or branching. This simple program structure contributes to our ability to implement IIR.

We are counting on these surfaces to intersect with each other in ways we cannot predict. Each Boolean operation we apply—intersection, union, or subtraction—forms a new enclosed surface, which may be assigned its own architectural function.

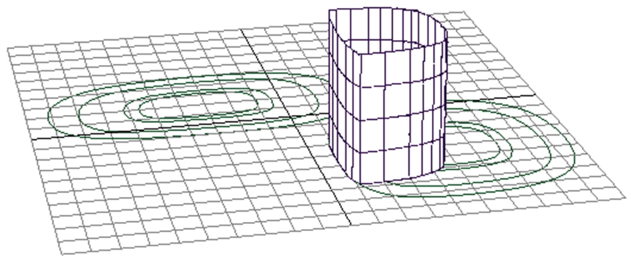
Fig. 7. Boolean Intersection

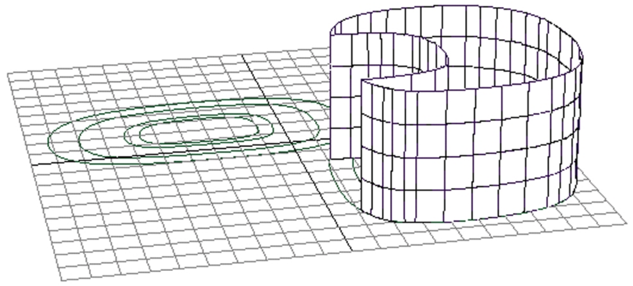
Fig. 8. Boolean Subtraction

Each individual in a population, therefore, is comprised of several separate Maya/C++ objects each of which carries a NURBS surface, certain evolvable values, and a list of operations. That we never directly query or modify the low-level geometry of the Maya objects, but allow Maya to perform all needed transformations and Boolean operations is what makes the language high enough level to be useful for the direct intervention of a designer.

Interruption, Intervention, and Resumption

Once a population has been ranked by fitness, the AGENCY GP becomes open to IIR. The entire population of interpreted designs is available for viewing by the user, who has several options. It is possible simply to re-rank individuals and allow evolution to continue, or to take a candidate design and apply one or more operations from our language to an arbitrary number of the NURBS surfaces that comprise it. The transformations applied will be added to the list of operations in the internal representation of the individual. By providing the basic operations of three-dimensional modeling through our language we enable designers to make targeted modifications of designs before allowing evolution to continue.

Layers of User Interactivity

IIR and specifying the seed design are the most direct modes of user interaction with the system. However, there are several other layers of user interactivity built into AGENCY GP. A designer is free to control what and how many pre-existing agents to deploy to evaluate designs, or how heavily to weight each agent's findings. Agents may also have controls of their own which allow a user to direct their activities. We have also specified and documented several C++ interfaces so that users may write their own modules for inclusion into AGENCY GP. Users are encouraged, for instance, to write agents with concerns specific to their use of AGENCY GP.

Future Work

We are currently in the process of implementing agents for the design of office spaces. This in itself implies a program of research, which will determine what are fruitful metrics by which to analyze designs. We are certain to learn through study and experiment the capabilities and limitations of our approach. Once we have created a body of agents, the AGENCY GP itself becomes a platform for research. We will be able to determine, for instance, what agents are difficult to satisfy simultaneously. By looking at patterns of IIR and users' preferencing of agents, we can also determine which agents are working at cross-purposes with human designers, and thereby draw inferences about the nature of our own preferences.

Acknowledgements

Jason Rolfe
Ian Fergusson
Nina Chen
Omar Khan

Sources:

Hyacinth S. Nwana, "Software Agents: An Overview," *Knowledge Engineering Review*, Vol. 11, No 3, (Sept 1996): 1-40.
Dejan Milojicic, "Mobile Agent Applications," *IEEE Concurrency*, Vol. 7, No. 3, (July-Sept 1999).
John R. Koza, Genetic Programming: On the Programming of Computers by Means Of Natural Selection (Cambridge: MIT Press, 1992).