Architectural Genomics

By Keith Besserud, AIA and Josh Ingram
Skidmore, Owings, & Merrill LLP (BlackBox Studio)

Abstract

This paper provides an introduction to the concept of genetic algorithms and a sampling of how they are being explored as an optimization strategy for some of the building projects in the BlackBox Studio at Skidmore, Owings, & Merrill LLP.

Biological Origins

The genome of any organism is its biological transcription. It is the identification, in order, of the sequence of nucleotides which are the “ladder rungs” of its DNA strands. In the case of the human genome there are just 4 nucleotide types (transcribed as A, C, G, or T) which are irregularly combined over and over into a string which is about 3 billion characters long and which determines every physiological trait of each individual. It is the similarities and differences of the sequences of these nucleotides from one person to the next which explains the similarities and differences between those two people. It is the fact that a portion of an individual’s genome comes from the mother and a portion from the father which causes that individual to resemble one parent or the other in certain biological respects. Without the genome there is no such thing as generational inheritance and without the genome there would be no such thing as evolution, at least not in the progressive sense with which we are familiar.

Because of the genome, because of inheritance, and because of the transformations of the genome from one generation to the next, natural evolution has provided a means of optimizing the fitness of its creatures. The result of this process is often referred to as “the survival of the fittest”.

Although the fields of genomics and genetics are concerned with only living organisms (since only living organisms have genomes) the phenomenon of procreation is central to these interests and so creation, in all its manifestations (including architectural design) can potentially be contemplated within the parameters of these fields of thought. In other words, it may be worthwhile to consider architecture and buildings as genetic organisms, able to be described by sequences of characters that encode the expression of their forms and the performance of their systems. This is the basis for the introduction of the concept of the genetic algorithm (GA) into the field of architectural design.

Based on theories of natural evolution, a genetic algorithm relies on a combination of randomness methods and goal-driven methods to search out the fittest members of a given population. Genetic algorithms work well for searches in which there is no preconception of the design solution and the field of possible design solutions (the solution space) is far too vast to accommodate the fitness testing of every single possibility in a timely manner. So instead of testing every single solution, the genetic algorithm relies on an ability to correlate a solution’s performance to its genome. The algorithm basically assumes that if a particular

genome achieves a strong fitness score, then other genomes which are similar to it or are derived from it might perform even better. So as not to just settle for the best performer within a certain “neighborhood” of the entire solution space, the genetic algorithm continues to try to generate genomes which are located in all corners and pockets of the solution space, even as it is simultaneously working to pinpoint the best individual in a particular neighborhood.

Design of GA’s

At SOM we have been developing and using genetic algorithms as a means of searching for optimal design solutions for various design problems and fitness criteria, and also as a means of developing a better understanding of the nature of the design of the GAs themselves. What we have found as we have been working with the design studios is that we are not necessarily interested in finding only the single most optimal design solution (in fact, although GA’s are a very good search methodology, they are not guaranteed to find the absolute optimal), but rather we are interested in being able to visualize and evaluate a range of many well-performing design solutions. The intent is not to completely automate a process of design, but to quickly and effectively generate better information to assist the design team in making design decisions.

At a conceptual level, the recipe for a GA is relatively simple:

1) Define the fitness function(s)…what performative metric(s) is(are) being optimized?
2) Define a genome logic (the number of characters in the genome string and the relationship between the individual characters and the expression of the design geometry)
3) Randomly generate an initial population of genomes
4) Test the fitness of the designs that are generated from each genome
5) Identify the top performers; these will become the selection pool for the next generation
6) From the selection pool build the population of the next generation, using methods of cloning, cross-breeding, mutation, and migration
7) Test all the new genomes in the new population
8) If the performance appears to be converging to an optimal condition then stop; otherwise repeat starting from step #5

Within this relatively simple recipe are implied a very large number of variables and relationships which must be defined and which directly affect the efficiency and results of the GA. What is the size of the population? What is the length of the genome string? How do the genome characters control the design and allow it to change in sufficiently interesting and useful ways? How is fitness defined and measured? How are multi-objective scenarios reconciled? How are the genomes chosen that become the selection pool for the next generation? How are the genomes of the next generation actually generated (huge number of variables)? How is convergence defined and detected?

Different answers to these questions will yield different behaviors from the GA, however, all GA optimization processes can be understood fundamentally in terms of 3 mandatory components: the fitness function; the genome; and the evolutional engine.

The Fitness Function

If the primary purpose of a genetic algorithm is to find optimal solutions, then the first question that must be answered is: “What is the phenomenon or quality that is being optimized?” For architectural design problems there are all kinds of possibilities. Some of the more obvious categories include construction cost, structural efficiency, carbon footprint, daylighting quality, acoustic quality, programmatic compliance, view quality, etc. Basically any parameter that exerts an influence on the execution of the design intent is eligible to become the metric of the fitness function.

There are a few technical issues to keep in mind when considering a fitness function. First of all there must be a means of simulating the phenomenon or quality that is being evaluated, and the results of the simulation must be numerically quantifiable. Also, the processes of simulating and quantifying must be automatable, either in an existing piece of software or a custom written script. And if (as is usually the case with architectural problems) there is a geometric model onto which the simulation is being executed, there must be an automated process for exporting/ importing the geometry into the simulation environment or there must be a means of exporting/ importing the coordinate data and then building the geometry directly within the simulation environment. On the backside, the fitness function simply needs to be able to output a value that is a measurement of the performance results of the simulation, which will then serve as an input to the evolutional engine.

Often we need to track more than just one criteria as part of our evaluations (including criteria that might even be in opposition to each other), and these “multiple objective” optimization problems can be dealt with in a few different ways, including the use of penalty functions or the incorporation of Pareto algorithms in order to generate multiple possible solutions. Thus far in our explorations, we have only developed experience with penalty functions. In the penalty function approach to working with multiple objectives, a primary fitness function performance is awarded a score and then that score is modified by one or more penalty factors which are derived from the design’s performance with respect to the other fitness functions. So, for example, if the challenge is to maximize surface radiation on the façade of a building in order to deploy PV panels, and also to maintain a specific target floor area for the entire building, then the solar radiation performance might establish the initial score, which could then be reduced or “penalized” according to how close the design comes to the target floor area.

The principal challenge with penalties, in our experience, is to calibrate the effect of the penalty. In the previous example, if there is a high priority on maximizing the solar radiation and the square footage target is somewhat less imperative, then conceivably the design team might be interested in a solution that scores very highly on radiation, even though it exceeds the area target by 5%. In this case the penalty should not be so severe that this genome's final score causes it to be pushed out of the pool of top performers in the population.

One last important consideration about the fitness function is computation time. It is not uncommon for a GA to have to run tens of thousands of iterations to reach convergence. Even if it takes just a few seconds to complete a simulation and analysis iteration, the total optimization process could take many days to reach convergence. The speed of the fitness function is the single most influential factor in the efficiency of the GA (assuming the problem is set up properly to be able to reach convergence).

The Genome

Similar to the human genome, the purpose of any particular genome in a GA is to encode and preserve the instructions which determine the form of a specific design. With a genome string that is 3 billion characters long, the human genome is capable of generating an enormous amount of diversity and nuance. For a typical architectural problem, having a genome of 3 billion characters available to describe the geometry would afford great flexibility and fidelity, however, the time required to compute an optimization to the point of convergence with such a long genome would usually be well beyond the time that is available for most projects.

The genomes have to be much smaller for our purposes. Depending on the nature of the formal design intent, our genomes have typically been anywhere from 5 to 70 characters long. The range of possible values for each character of the genome is also a consideration. For our purposes we have been using integer values from 0 to 99, but the implication of these choices is that the size of the genomic matrix (the number of characters * the number of possible values for each character) is also a determinant in the efficiency of the optimization process. As may be obvious, the smaller the genome string, the fewer the number of generations required to reach convergence, but the less rigorous the level of control over the geometry. For the types of problems we have been studying we are typically using genome values to identify point coordinate values, but the genome characters could also be used to define other things, such as variable values in polynomial formulas that define curves or surfaces.

No matter how many characters there are in the genome string there will be a set of forms that is derivable and a set that is not. Because of this it is important for the design team to put some thought into the relationship between the genome

and the geometry it is controlling in order to anticipate the types of forms that will be developable based on the genome logic.

The Evolutional Engine

The evolutional engine is the script that turns the crank to keep the optimization process moving along. Its major functions are to feed genome strings (or building geometries generated from the genome, depending on how the geometry is being generated) to the fitness function, to take analysis results from the fitness function, to rank the genomes of each generation, to identify the selection pool for the next generation of genomes, and to generate new populations of genomes for feeding into the fitness function. This script keeps this process going over and over for as many generations of genomes as is needed to reach some definition of convergence.

The evolution engine we have been developing is intended to be somewhat generic, allowing for greater flexibility. It has been written so that it can be coupled with any fitness function and any definition of a genome string (with minimal editing), and it has a user interface that allows the user to custom set a wide range of variable values for controlling the initial population and the methods for generating each successive population.

The point of exposing so many of the GA design variables in a user form is to assist us in the process of understanding the behavior and performance of the GA. By manipulating the variables and studying the results of many optimization runs we are beginning to understand the more sensitive variables and some of the more effective settings. In general a couple of things have begun to seem evident.

First, it seems to be very important to generate and maintain as much diversity as possible in every population. Obviously the development of diversity means the GA is probably doing a better job of hunting down obscure, but well-performing genomes. But the diversity also is important to the cross-breeding and mutation processes to prevent patterns of in-breeding from taking over and dominating the population.

Another observation has to do with the shape of the plot curve that traces the score of the top performer in each generation (see Figure 2). In general these curves start out quite steep and then gradually flatten out, meaning performance improves at a fast pace early on, but then becomes very incremental. The incremental growth curve is indicative of movement within the neighborhood of a local optimum, whereas the steep climb suggests that the GA is jumping effectively around the design space.

As mentioned earlier, we are often not so interested in finding the absolute optimal solution, so the determination of how many generations to run the GA is

rather subjective. Attaining an improvement of 20% in the first 100 generations is not uncommon, but the next 10% may take 300 more generations and may simply be the result of modest tweaks to the genome values, and so the benefit, at least at a conceptual stage of design, may be debatable. The time might be better spent setting up new runs of the GA to possibly uncover solutions in completely different areas of the design space.

Project Example – Optimization of a Hypothetical Tower

For this exercise the objective was to find the optimal shape for a hypothetical 300-meter tower (subject to a handful of geometric constraints) in order to maximize incident solar radiation on the skin of the building. The working assumption was that this form would best suit the deployment of photovoltaic panels to collect solar radiation.

The geometry of the tower was controlled by a combination of length and point coordinate parameters (see Figure 1). Plan profiles at 6 different locations up the height of the tower were used as guides to sweep the tower skin. The plan profile at the base of the tower was fixed in its location and dimensions. At each of the other 5 plan profiles the shape was parametrically defined by a circular or elliptical polyline which was constrained by two values that controlled the major axis and minor axis lengths. The lengths were constrained to a defined range. In addition, the center point of the ellipse was allowed to move in the x and y directions, also within constrained limits.

Each of these 4 geometric parameters at each floor was controlled by a value in the genome string which defined either a length for an axis or the x and y coordinates of the center point. For all 5 floors which were defined parametrically there were a total of 20 characters in the genome string. In addition to these dimensional constraints, target values were established for the maximum floor area and maximum skin area.

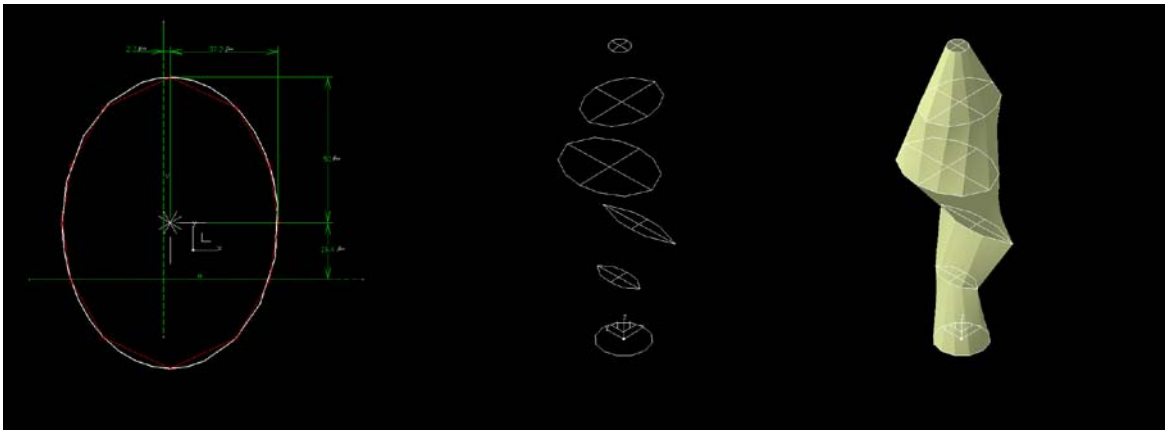

Figure 1: Tower genome consists of 4 characters for each of 5 parametric floor profiles.

The genetic algorithm went through a process of generating an initial population of 75 different genomes, resulting in 75 different tower forms, and then ran each tower through the solar analysis module in Ecotect to calculate the cumulative annual radiation incident on the surface of each tower. Additionally, the floor and skin areas were calculated. After penalties were assessed for floor and skin area deviations from the targets, the GA entered into an iterative routine of determining the selection pool of best performers, creating the next generation of genomes and towers through cross-breeding and mutation, running the solar analysis on each of the new towers, and then determining the new selection pool, etc.

We ran the GA for two different scenarios, using Las Vegas weather data for both. In the first scenario we isolated the tower with no context in order to avoid contextual shading and to see how the tower form would evolve based purely on the solar path and weather data. It was interesting to see that even though the sun moves in a symmetrical pattern over the course of a day, all year long, the optimal geometry that was generated was not a symmetrical form (see Figures 3, 4, 5).

In the second scenario we positioned hypothetical buildings around the test building such that shadows would be cast onto the test building at different times of the day and year (see Figure 9). In this scenario it was interesting to observe how the building worked to shape itself to find opportunities for greater sun exposure (see Figures 6, 7, 8).

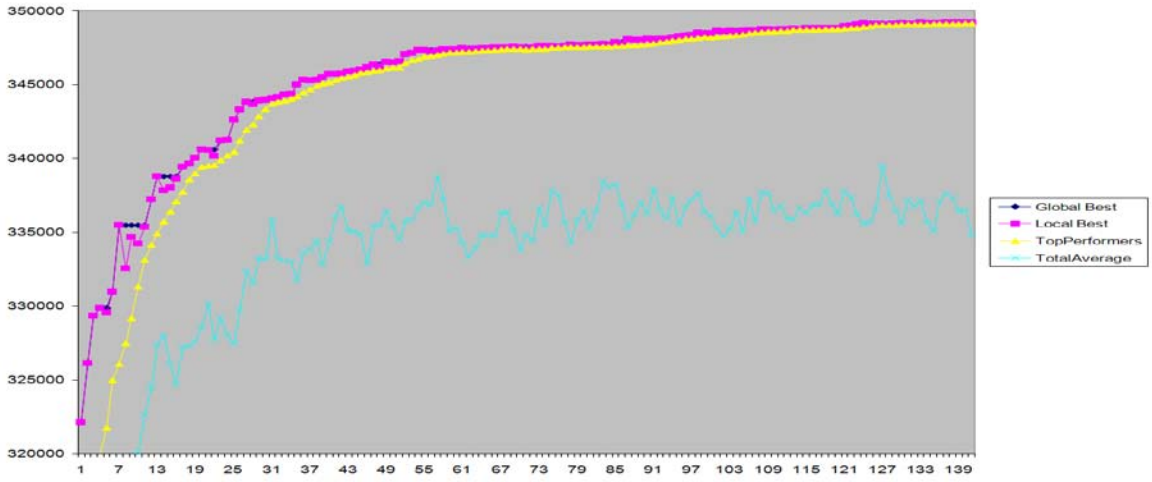

Figure 2: Sample results curve showing number of generations along horizontal axis and fitness function score along vertical axis. This graph shape is typical for the GAs that we have run.

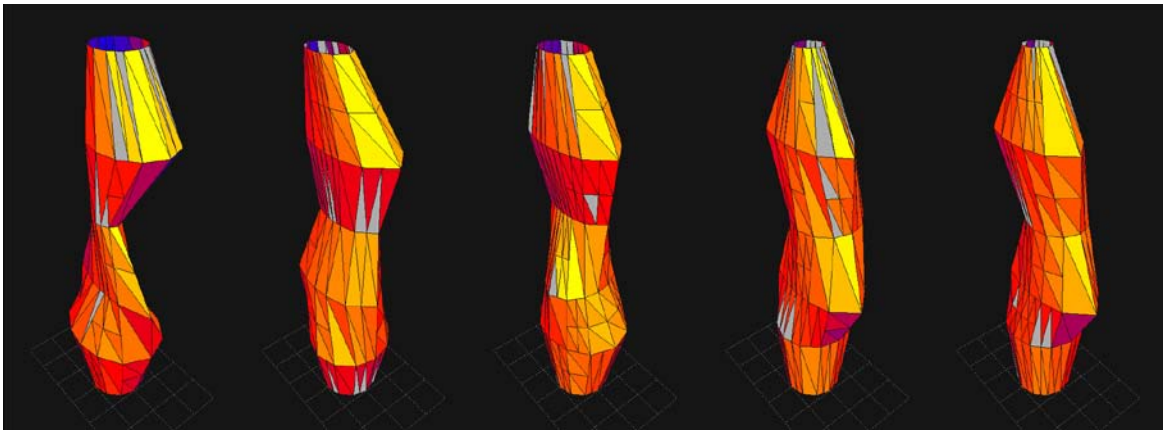

Figure 3: Evolution of tower form with no adjacent context; Generations 1, 5, 11, 31, and 101

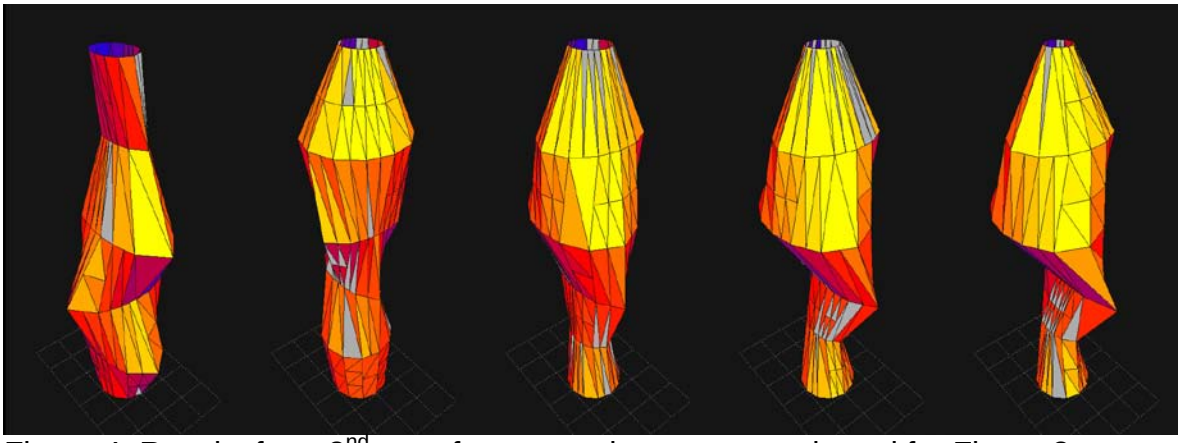

Figure 4: Results from 2nd run of same study as was conducted for Figure 3.

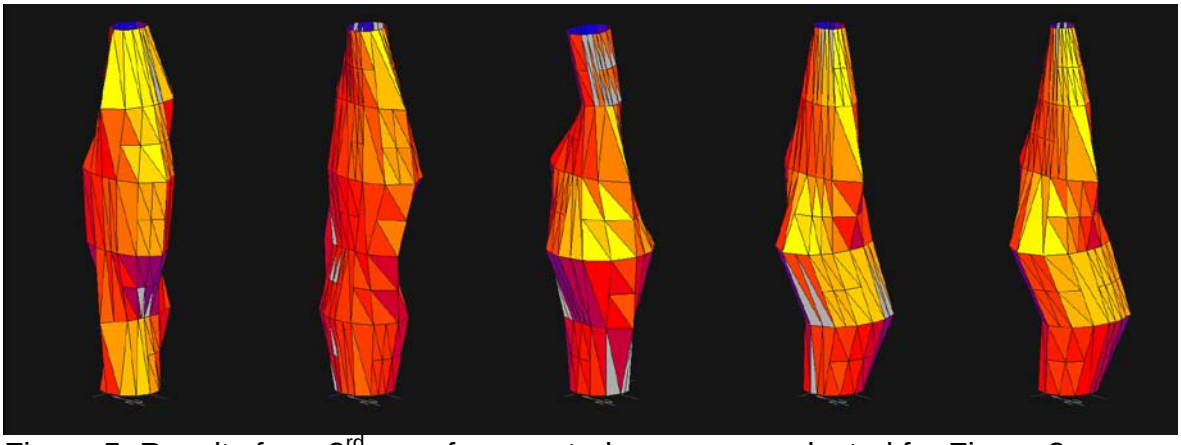

Figure 5: Results from 3rd run of same study as was conducted for Figure 3.

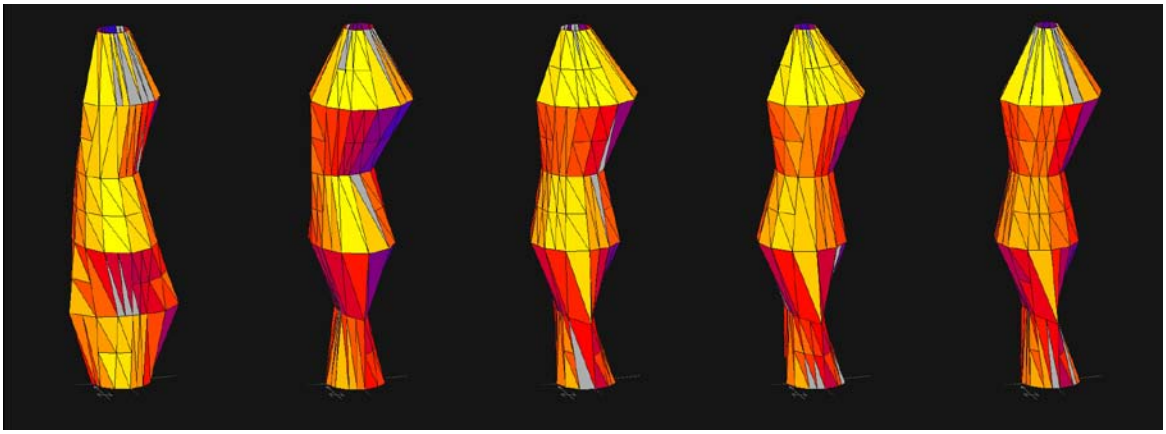

Figure 6: Evolution of tower with adjacent contextual shading; Generations 1, 5, 11, 31, and 101

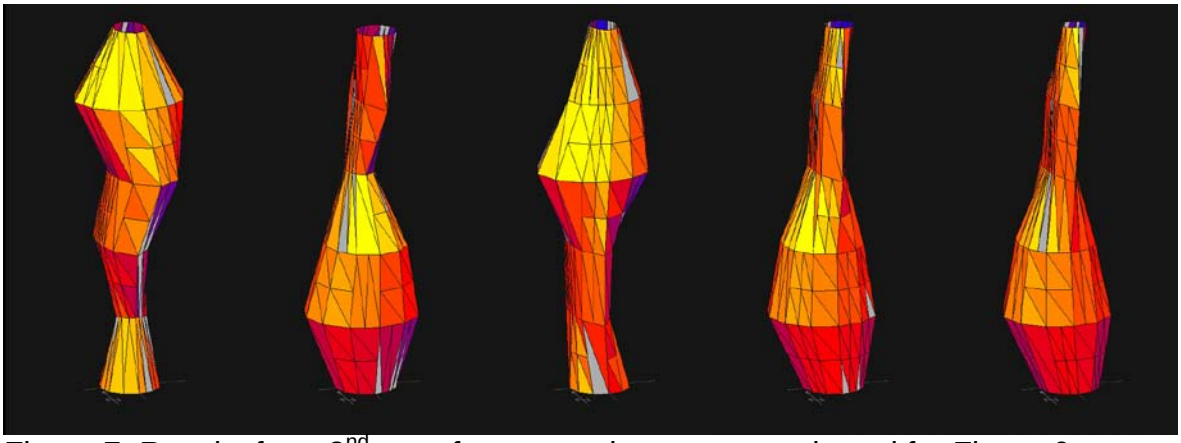

Figure 7: Results from 2nd run of same study as was conducted for Figure 6.

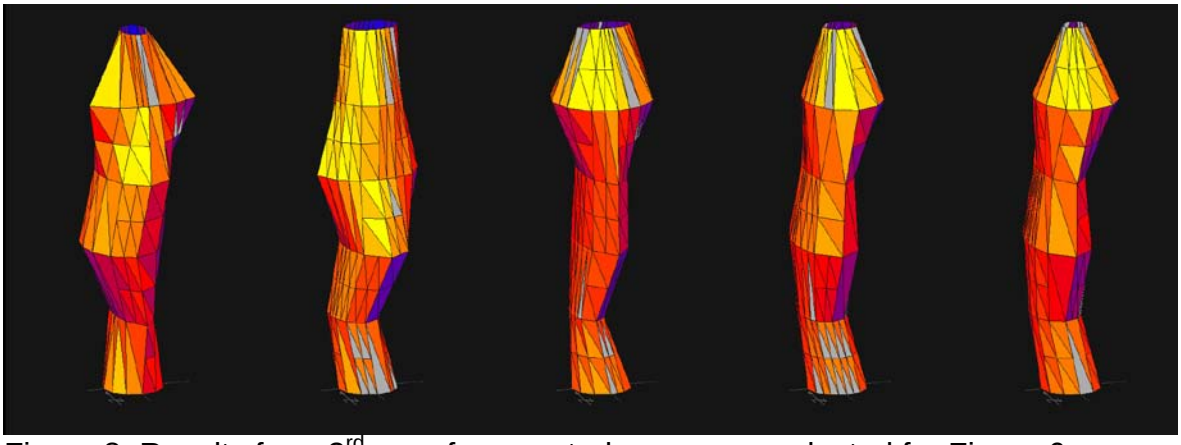

Figure 8: Results from 3rd run of same study as was conducted for Figure 6.

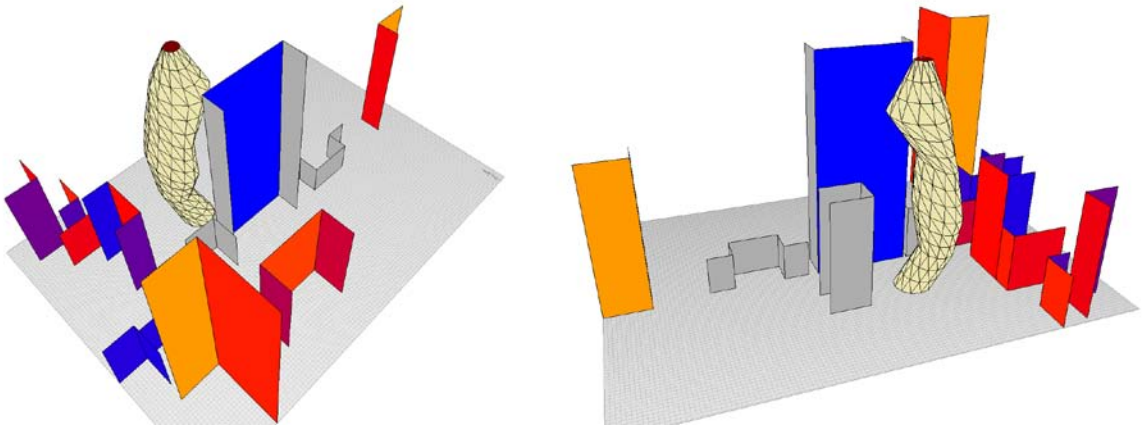

Figure 9: Views of tower from Figure 8 positioned within context

Conclusions & Further Research

Optimizations that were run 3 separate times for each scenario (with and without context) yielded a variety of convergence positions. In the 3 runs with no context the amount of surface radiation from the first generation to the final generation increased by an average of about 6% and in the 3 runs with context the gain was about 12%. Compared with a “base case” geometry of a simple cylinder of the same height and surface area, the context-free solutions were about 21% better, and the context-based optimization was about 27% better. In the most fundamental of goals, the GA did work to find optimal results.

It does not appear, however, that in this experiment we can say with certainty that we found the global optimal solution for either scenario. Instead we can only say that it appears we found a number of local optima. Had the GA been able to find the global optimum then we should have seen each of the 3 runs converge on the same form. The reason for the multiple convergences we observed may simply be that there are a lot of different building forms that all perform about the same in terms of harvesting solar radiation, and so there is no single, disproportionally high peak within the search space. In the scenario with no context, it seems quite probable that this might be the case.

In the scenario with contextual shading we actually had somewhat similar convergences for 2 of the 3 runs, which suggests that the numbers of possible peaks in the search space was fewer. This makes sense because the added constraints of adjacent buildings meant there were fewer openings for sunlight to get through and so the test building had fewer configurations available that would allow it to efficiently harvest available sunlight.

Another possible explanation for the multiple unique convergences may be in the design of our GA, and to that end we are interested in exploring a number of areas of focus in order to develop improvements in the design of the algorithm. The first is to look at the design of the genome string and to try to find better ways of being able to control greater levels of complexity in the geometric models

with smaller genome strings. We see the use of polynomial formulas as one possible approach. Another might be to use the genome characters to refer to various function ID's in which any number of operations might take place, possibly even leading to the use of directed graphs in some way.

Secondly, we would like to investigate the concept of adaptive methodologies. In this approach, methods of population-sizing, searching, selection, mutation and cross-breeding might be adjusted during the course of running the GA. For example, mutation operations at the early stages of the GA might affect a higher number of genes and to a larger variation, then gradually adjust to affect fewer genes and to a tighter range during later stages of the run when it has entered a hill climbing mode. Or, as the GA is settling into an optimal neighborhood, the searching methodology might transition from the classic GA approach to a swarm optimization technique or some gradient-based optimization strategy that would speed the convergence process.

A third trajectory of investigation would involve improving the ability of the algorithm to output and save optimal performing designs from multiple peaks. Currently, as the GA approaches convergence, the top performers are usually from the same neighborhood and so there is probably not much diversity in the forms of the designs. Instead what would be preferable would be to simultaneously develop forms from many different neighborhoods in the solution space which look very different from one another, but which are all pretty good performers with respect to the performance goals.

Finally, and perhaps most compelling, we intend to look into the concept of interactive subjective GA's. In this scenario, the designer who is directing the GA would have the ability at the conclusion of each generation to survey the forms that were generated and make preferential selections, irrespective of performance. The GA would then factor those choices into the way it scores performance such that subsequent "top performers" would be biased according to some reconciliation between the original performative criteria of the fitness function and the new subjective criteria provided by the user.

Credits

As this paper is derived from a process that relied more on empirical development and testing than on theoretical underpinnings, credits are more general in nature. The concept of the genetic algorithm is not novel, but it is a new frontier in the Chicago office of SOM and although the computer code we generated is entirely proprietary, we did benefit along the way from numerous valuable conversations with 3 people in particular: Victor Keto, with Gehry Technologies; Dr. David Goldberg, with the Illinois Genetic Algorithms Lab; and Dr. Una-May O'Reilly, with MIT.