UDC 005.922.52:004.932, DOI: 10.2298/CSIS1001111L

Generative 3D Images in a visual evolutionary computing system

Hong Liu

School of Information Science and Engineering, Shandong Normal University, Jinan
lhsdcn@jn-public.sd.cninfo.net

Abstract. This paper presents a novel computer-aided design system which uses a computational approach to producing 3D images for stimulating creativity of designers. It introduces the genetic algorithm first. Then a binary tree based genetic algorithm is presented. This approach is illustrated by a 3D image generative example, which uses complex function expressions as chromosomes to form a binary tree, and all genetic operations are performed on the binary tree. Corresponding complex functions are processed by MATLAB software to form 3D images of artistic flowers. This generative design is integrated with a visualization interface, which allows designers to interact and select from instances for design evolution. It shows the system is able to enhance the possibility of discovering various potential design solutions.

KeyWords: Evolutionary computation, computer-aided design, generative design, complex function

1. Introduction

To date, more and more computers have been used in design offices. The CAD (computer-aided drafting or computer-aided design) tools frequently used in these design offices provide two types of support. These tools are for two-dimensional (2D) drafting and three-dimensional (3D) modeling tasks. The drafting tools replace traditional tools such as pencils and rulers to produce detailed design drawings, while modeling tools take over the functions to allow the visualization of new designs. These tools assist designers in the final production and presentation of the design products.

However, few of those CAD tools have been used to assist designers in the early phases, such as the conceptual design process. During the early phases, designers explore many design alternatives. Current CAD tools do not provide sufficient design support to innovative design.

In cognitive psychology, design activities are described as specific problem-solving situations, since design problems are both ill defined and open-ended. Design activities, especially in ‘non-routine activities’, designers involve a special thinking process. This process includes not only thinking with logic, but also thinking with mental imagery and sudden inspiration.

Designers have called new ideas in their mind as idea sketches. In contrast to presentation sketches, idea sketches are made in the early phases of design. They function as a tool to interact with imagery and are predominantly for private use. Because of their early appearance in the design process, idea sketching will have an important role in creative processes. This is the reason why many computer tools aim at supporting and improving idea sketching.

Creative ideas occur in a particular medium. Most of the researchers in the field of creativity agree that designers who are engaged in creative design tasks use external resources extensively. Such external resources include a variety of physical and logical information. For instance, reading books, browsing photographic images, talking to other people, listening to music, looking at the sea or taking a walk in the mountains. Sketches and other forms of external representations produced in the course of design are also a type of external resources that designers depend on. When designers discover a new or previously hidden association between a certain piece of information and what they want to design, the moment of creative brainwave emerges. Designers then apply the association to their design and produce an innovative design.

Visual images are particularly for activating creativity. In product design, visual expression, especially in the form of sketching, is a key activity in the process of originating new ideas. This approach suffers from the fact that most creative processes extensively make use of visual thinking, or, in other words, there is a strong contribution of visual imagery. These processes are not accessible to direct verbalization.

The design research community has spent much of its effort in recent years developing design systems for supporting innovative design. Generative design systems - systems for specifying, generating and exploring spaces of designs and design alternatives - have been proposed and studied as a topic of design research for many years.

Generative design is an excellent snapshot of the innovative process from conceptual framework through to specific production techniques and methods. It is ideal for aspiring designers and artists working in the field of computational media, especially those who are interested in the potential of generative, algorithmic, combinational, emergent and visual methods as well as the exploration of active images.

This paper presents a novel computer supported design system that uses the evolutionary approach to generate 3D images. The tree structure based genetic algorithms and complex functions are used in this system. This generative design is integrated with a visualization interface, which allows designers to interact and select from instances for design evolution. Programs are implemented by using Visual C++6.0 and mathematical software MATLAB.

The remainder of this paper is organized as follows. Section 2 is concerned with related work on generative design. Section 3 introduces genetic algorithms and genetic programming. In section 4, a binary tree based genetic algorithm is presented. Section 5 illustrates an artwork design example for showing how to use the tree structure based genetic algorithm and complex

functions to generate 3D images. Finally, these results are briefly analyzed, followed by a discussion of possible future improvements.

2. Related work

Generative systems are relevant to contemporary design practice in a variety of ways. Their integration into the design process allows the development of novel design solutions, difficult or impossible to achieve via other methods. Grammar-based techniques exploit the principle of database amplification, generating complex forms and patterns from simple specifications. Evolutionary systems may be used in combination with aesthetic selection to breed design solutions under the direction of a designer. Interface design and other sign systems may be defined in terms of adaptive procedures to create communication that adapts to its interpretation and use by an audience [1].

The key properties of generative systems can be summarized as [2]:

• The ability to generate complexity, many orders of magnitude greater than their specification. This is commonly referred to as database amplification, whereby interacting components of a given complexity generate aggregates of far greater behavioral and/or structural complexity. Such aggregates may in turn generate their own interactions forming new aggregates of even higher sophistication and complexity. This is referred to as a dynamic hierarchy, a poignant example being complex multi-cellular organisms, whose hierarchy can be summarized: atom; molecule; organelle; cell; organ; organism; ecosystem.

• The ability to generate novel structures, behaviors, outcomes or relationships. Novelty used in this sense means the quality of being new, original and different from anything else before it. There are of course, different degrees of novelty. RNA and DNA was novel in that they introduced a completely new mechanism for replication and encoding of protein synthesis. Artists and designers are always seeking novelty (the opposite of which is mimicry or copying, something depreciated in the art and design world). Artistic novelty may not have such a significant impact as, for example, DNA, but the key concept is that of the new — generative systems have the potential to give rise to genuinely new properties. This is why they are often referred to as emergent systems. These new properties typically fall outside the designer's expectations or conceptualizations for the design, resulting in functionality or outcomes that were not anticipated. This of course raises the issue of control, a problematic issue for generative design, particularly if the designer is accustomed to organizing outcomes in a predictable way.

Generative design describes a broad class of design where the design instances are created automatically from a high-level specification. Most often, the underlying mechanisms for generating the design instances in some way model biological processes: evolutionary genetics, cellular growth, etc. These artificial simulations of life processes provide a good conceptual basis for designing products.

Evolutionary systems are based on simulating the process of natural selection and reproduction on a computer. This technique has found wide application in design for computer animation and graphics, but also in architectural, industrial and engineering design [3]. The technique depends on the specification of a parameterized model that is general enough to allow a wide variety of possible outcomes of interest to the designer. In cases where a very specific goal is sought, the parameter space must also be sufficiently broad to encapsulate an answer to the problem that meets the specified design constraints.

Initially a population of potential designs is generated with a random set of parameters. This random population may be displayed visually to the designer. The designer's aesthetic sense then determines the 'fittest' designs of those displayed, and these are 'bred' with one another to produce a new population of designs that inherit the traits of their successful parents. This process is akin to selective breeding of apple trees for the taste and color of their fruit – both subjective qualities assessed by humans [4].

An alternative means of utilizing this evolutionary process exists where a fitness function may be explicitly coded by the designer. For example, perhaps a designer seeks the lightest, cheapest set of automobile wheels. This fitness function is coded into the evolutionary system and the computer evolves populations of wheels towards a successful structure independently of further human input. Where creative designs are sought with some aesthetic value, the former technique of interactive evolution is more practical. This requires constant human interaction, a necessary bottleneck as long as it remains difficult to encode subjective qualities like 'beauty', 'ugliness' etc. in such a way that a computer can operate with them.

Some of the evolutionary design work was performed by Professor John Frazer, who spent many years developing evolutionary architecture systems with his students. He showed how evolution could generate many surprising and inspirational architectural forms, and how novel and useful structures could be evolved [5]. In Australia, the work of Professor John Gero and his colleagues also investigated the use of evolution to generate new architectural forms. This work concentrates on the use of evolution of new floor plans for buildings, showing over many years of research how explorative evolution can create novel floor plans that satisfy many fuzzy constraints and objectives [6]. They even show how evolution can learn to create buildings in the style of well-known architects.

In Argenia, a system for architectural design by Soddu, the three-dimensional models produced can be directly utilized by industrial manufacturing equipment like numerically controlled machines and robots, which already represent the present technologies of industrial production. This generative and automatic reprogramming device of robots makes it possible to produce unique objects with the same equipment and with costs comparable to those of objects that are identical; like a printer that can produce pages that are all the same or all different, at precisely the same cost [7].

Many artists and designers have turned to generative systems to form their design basis. Dextro (www.dextro.org) has developed a diverse range of

interactive drawing systems in which simple design elements such as points and lines replicate and self-organize to create illustrations and animations. Digital artists Meta use generative processes to create streams of abstract video (www.meta.am), expressive of the multi-layered, fluid mutable nature of electronic space. Jared Tarbell (www.levitated.net) has experimented with the intersection of generative systems, typography and graphic patterns in his experimental web design projects. Software such as Auto-Illustrator (www.auto-illustrator.com) and Autoshop (www.signwave.co.uk) combine generative systems with the image composition and editing functions of popular computer drawing programs to 'automatically' create new designs ready for use in projects. Groboto (www.groboto.com), a program that allows users to develop their own systems for growing 3D forms, makes these systems accessible to a wider audience by placing a GUI in front of the lines of code usually required to work with generative systems.

However, the development of generative design tools is still at its early stage. The research and development of design support tools using evolutionary computing technology are still in process and have huge potential for the development of new design technology.

3. Genetic algorithms and genetic programming

General GA (genetic algorithm) is a search algorithm based on the mechanisms of natural selection. They lie on one of the most important principles of Darwin: survival of the fittest. John Holland, in the 1970s, thought that he could incorporate in a computer algorithm such a technique, to solve different problems through evolution [8].

Given a specific problem to solve, the input to the GA is a set of potential solutions to that problem, encoded in some fashion, and a metric called a fitness function that allows each candidate to be quantitatively evaluated. These candidates may be solutions already known to work, with the aim of the GA being to improve them, but more often they are generated at random.

The GA then evaluates each candidate according to the fitness function. In a pool of randomly generated candidates, of course, most will not work at all, and these will be deleted. However, purely by chance, a few may hold promise - they may show activity, even if only weak and imperfect activity, toward solving the problem.

These promising candidates are kept and allowed to reproduce. Multiple copies are made of them, but the copies are not perfect; random changes are introduced during the copying process. These digital offspring then go on to the next generation, forming a new pool of candidate solutions, and are subjected to a second round of fitness evaluation. Those candidate solutions which were worsened, or made no better, by the changes to their code are again deleted; but again, purely by chance, the random variations introduced into the population may have improved some individuals, making them into better, more complete or more efficient solutions to the problem at hand. Again

these winning individuals are selected and copied over into the next generation with random changes, and the process repeats. The expectation is that the average fitness of the population will increase each round, and so by repeating this process for hundreds or thousands of rounds, very good solutions to the problem can be discovered.

Genetic algorithms have been gained recognition as solution techniques for many complex optimization problems [9], and have been applied to numerous conceptual and preliminary design studies [10, 11]. A genetic algorithm is not truly an optimization technique; rather it is a computational representation of processes involved in natural selection as observed in biological populations. They resemble natural evolution more closely than many other approaches because they are based on the mechanics of natural selection and natural genetics.

The GA consists of a number of elements. Once a problem has been identified, the elements can be broken down as follows:

- A representation of a potential solution;
- A population of chromosomes;
- A fitness function for evaluating the relative merit of a chromosome;
- A selection method;
- One or more operations for modifying the selected chromosomes (typically crossover and mutation);

These elements are then employed in an iterative process until a solution is found or a termination condition is met. An abstraction of a typical GA is given as follows:

Generate initial population, G(0);
Evaluate G(0); (apply fitness function)
t=1;
Repeat
  Generate G(t) using G(t-1); (apply operators)
  Evaluate (decode(G(t)));
  t=t+1;
Until solution is found or termination.

A search algorithm balances the need for exploration -- to avoid local optima, with exploitation -- to converge on the optima. Genetic algorithms dynamically balance exploration versus exploitation through the recombination and selection operators respectively. With the operators as defined earlier, the schema theorem proves that relatively short, low-order, above average schema are expected to get an exponentially increasing number of trials or copies in subsequent generations [12]. Mathematically

$$m(h,t+1) \geq \frac{m(h,t)f(h)}{f_t}\left[1 - P_c \frac{\delta(h)}{l-1} - \mathrm{O}(h)P_m\right]$$

Here m(h,t) is the expected number of schemas h at generations t, f(h) is the fitness of schema h and f_t is the average fitness at generation t. The genotype length is l, $\delta(h)$ is the defining length and O(h) is the order of

schema h. P_c and P_m are the probabilities of crossover and mutation respectively.

The schema theorem leads to a hypothesis about the way genetic algorithms work.

Genetic algorithms traditionally use string-based representations in their chromosomes, but this is not always suitable for representing higher level knowledge. Koza introduced a hierarchical GA approach [13], where his chromosomes are tree-like expressions that can be recombined by swapping sub-trees. It starts with a randomly generated population of function-trees, the trees which perform best on the problem in question are selected to be the breeding stock. These parent trees are combined (by exchanging sub-trees) and mutated (by generating new sub-trees) to produce new trees for the next generation of the population which inherit some features from their parents. The next generation of trees is then evaluated against the problem. The best trees are selected to produce the next generation after that, and so on.

Koza labeled his hierarchical version of GA, the Genetic Programming Paradigm (called GP). The main difference between traditional GA and GP lies in the following graph theoretical manipulation.

1. Mutation changing a node label, or substituting sub-trees.
2. Crossover i.e. swapping sub-trees

As with genetic algorithm, the GP population is evolved to (hopefully) produce function-trees which can perform well on the problem in question. GP has been used successfully in generating computer programs for solving a number of problems in a wide range of areas [14].

Nowadays, genetic programming is applied mostly related to adaptive system and optimization, where representation of programs is used in conjunction with hybrid crossover to evolve a multiplication function. In addition, the design with genetic programming is not traditionally considered in canonical genetic programming.

4. The tree structured genetic algorithm

General genetic algorithms use binary strings to express the problem. It has solved many problems successfully. But it would be inappropriate to express flexible problem. For example, mathematical expressions may be of arbitrary size and take a variety of forms. Thus, it would not be logical to code them as fixed length binary strings. Otherwise, the domain of search would be restricted and the resulting algorithm would be restricted and only be applicable to a specific problem rather than a general case. Thus, tree structure, a method useful for representing mathematical expressions and other flexible problems, is presented in this paper.

The main contribution of this paper is to use tree-like expressions as chromosomes for representing complex mathematical functions and apply the tree structured genetic algorithm in generative design.

For a thorough discussion about trees and their properties, see [15,16]. Here, we only make the definitions involved in our algorithm and these definitions are consistent with the basic definitions and operations of the general tree.

Definition 1 A binary complex function expression tree is a finite set of nodes that either is empty or consists of a root and two disjoint binary trees called the left sub-tree and the right sub-tree. Each node of the tree is either a terminal node (operand) or a primitive functional node (operator). Operand can be either a variable or a constant. Operator set includes the standard operators (+, -, *, /, ^) , basic mathematic functions (such as sqrt ( ), exp( ), log( )), triangle functions (such as sin(x), cos(x), tan(x),cot(x), sec(x),csc(x), asin(x),acos(x)), hyperbolic functions (such as sinh( ), cosh( ), tanh ( ), asinh ( ), acosh( ), atanh( )), complex functions (such as real(z), imag(z), abs(z), angle(z), conj(z)) and so on.

Here we use the expression of mathematical functions in MATLAB (mathematical software used in our system).

A binary complex function expression tree satisfies the definition of a general tree.

Genetic operations include crossover, mutation and selection. According to the above definition, the operations are described here. All of these operations take the tree as their operating object.

(1) Crossover

The primary reproductive operation is the crossover operation. The purpose of this is to create two new trees that contain 'genetic information' about the problem solution inherited from two 'successful' parents. A crossover node is randomly selected in each parent tree. The sub-tree below this node in the first parent tree is then swapped with the sub-tree below the crossover node in the other parent, thus creating two new offspring.

(2) Mutation

The mutation operation is used to enhance the diversity of trees in the new generation, thus opening up new areas of 'solution space'. It works by selecting a random node in a single parent and removing the sub-tree below it. A randomly generated sub-tree then replaces the removed sub-tree.

(3) Selection

For general design, we can get the requirement from designer and transfer it into goal function. Then, the fitness value can be obtained by calculating the similar degree between the goal and individual by a formula. However, for creative design, there are no standards to form a goal function. Therefore, it is difficult to calculate the fitness values by a formula. In our system, we use the method of interaction with the designer to obtain fitness values. The range of fitness values is from -1 to 1. After an evolutionary procedure, the fitness values appointed by designer are recorded in the knowledge base for reuse. Next time, when the same situation appears, the system will access them from the knowledge base [17].

This method gives the designer the authority to select their favored designs and thus guide the system to evolve the promising designs. Artificial selection

can be a useful means for dealing with ill-defined selection criteria, particularly user centered concerns.

Many explorative systems use human input to help guide evolution. Artists can completely take over the role of fitness function. Because evolution is guided by human selectors, the evolutionary algorithm does not have to be complex. Evolution is used more as a continuous novelty generator, not as an optimizer. The artist is likely to score designs highly inconsistently as he/she changes his/her mind about desirable features during evolution, so the continuous generation of new forms based on the fittest from the previous generation is essential. Consequently, an important element of the evolutionary algorithms used is non-convergence. If the populations of forms were ever to lose diversity and converge onto a single shape, the artist would be unable to explore any future forms.

For clarity, we will present the performing procedure of the tree structured genetic algorithms together with a flowers generative design example, in the next section.

5. A generative artwork example

An artwork design example is presented in this section for showing how to use tree structure based genetic algorithm and complex function expressions to generate 3D images.

The complex function expressions are used to produce 3D artistic images. Here, z=x+iy (x is real part and y is virtual part), complex function expression f(z) is an in-order traversal sequence by traversing complex function expression tree.

Both real, imaginary parts and the module of f(z) can generate 3D images in MATLAB. Three images of f(z)= cos(z)*log(-z^2)*angle(z^2) are shown as figure 1.

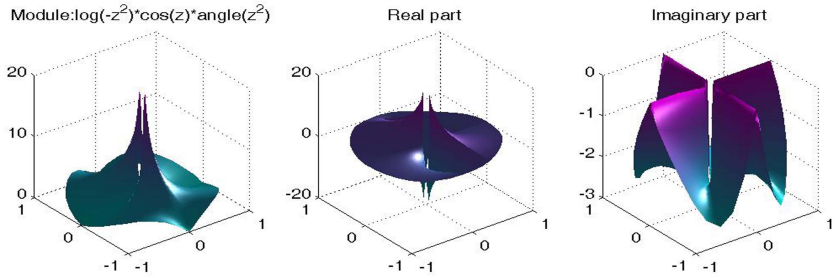

Fig. 1. Three images of f(z)= cos(z)*log(-z2)*angle(z2)

Next, we will present the performing process of the algorithm step by step.

Step 1: Initialize the population of chromosomes. The populations are generated by randomly selecting nodes in the set of operands and the set of operators to form complex function expressions. We use the stack to check whether such a complex function expression has properly balanced parentheses. Then, using parsing algorithm, the complex function expressions is read as a string of characters and the binary complex function expressions tree is constructed according to the rules of operator precedence.

Step 2: Get the fitness for each individual in the population via interaction with designer. The populations with high fitness will be shown in 3D form first. The designers can change the fitness value when they have seen the 3D images.

Step 3: Form a new population according to each individual's fitness.

Step 4: Perform crossover and mutation on the population.

Figure 2 shows two complex function expressions trees. Their expressions are $f(z)=\log(-z^2)*\cos(z)*\text{angle}(z^2)$ and $f(z)=\text{sqrt}(z)*\cos(-z^2)*\cot(-z*0.5)$ respectively.

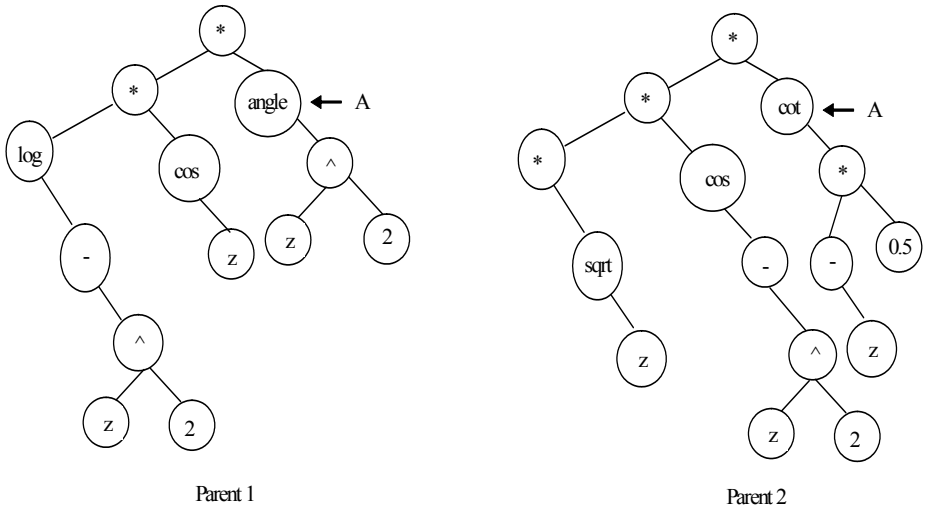

Fig. 2. Two parent trees with one crossover nodes

(1) Crossover operation

A crossover node is randomly selected in each parent tree. The sub-tree below this node on the first parent tree is then swapped with the sub-tree below the crossover node on the other parent, thus creating two new offspring. If the new tree can't pass the syntax check or its mathematical expression can't form a normal sketch shape, it will die.

Taking the two trees in figure 2 as parent, after the crossover operations by nodes 'A', we get a pair of children (figure 3).

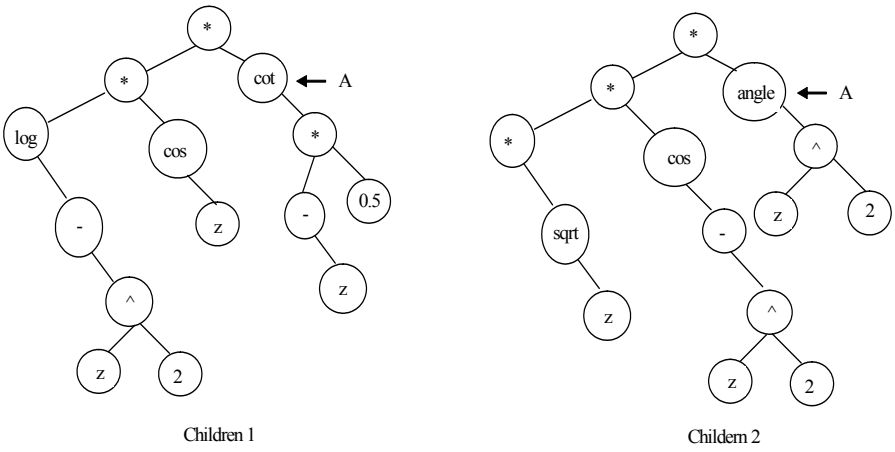

Fig. **3**. The two children generated by a crossover operation

Figure 4 shows a group of generated 3D images by the module part of f(z) correspond to figure 2 and figure 3.

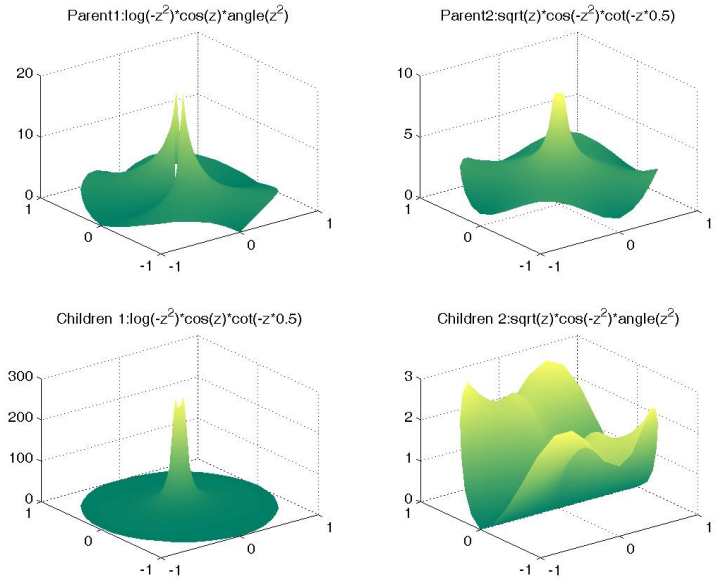

Fig. 4. The images correspond to the module of f(z) in figure 2 and figure 3

(2) Mutation operation

The mutation operation works by selecting a random node in a single parent and removing the sub-tree below it. A randomly generated sub-tree then replaces the removed sub-tree. The offspring will die if it can't pass the syntax check or it can't form a normal shape.

Taking $f(z)=\log(-z^2)*\text{angle}(z^2)$ as a parent, one offspring generated by mutation operation is shown as figure 5. In which, child is generated by replacing node A and its sub-tree with new sub-tree. Figure 6 is the two images correspond to the real part of f(z) in figure 5.

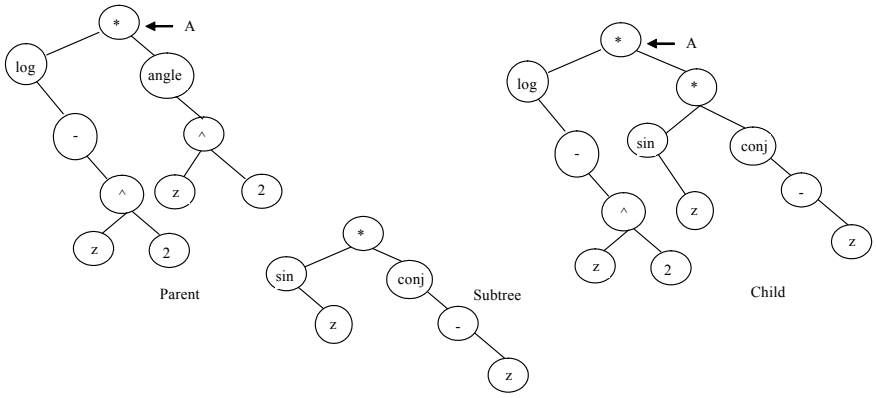

Fig. 5. One mutation operation

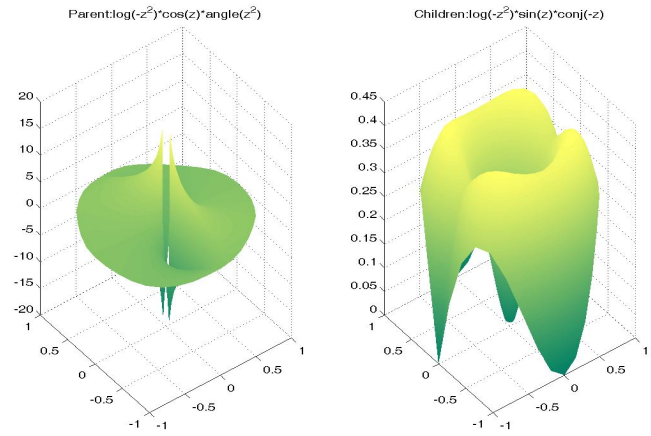

Fig. 6. The images correspond to the real part of f(z) in figure 5

Step 5: If the procedure is not stopped by the designers, go to step 2.

This process of selection and crossover, with infrequent mutation, continues for several generations until it is stopped by the designers. This procedural design is integrated with a visualization interface, which allows designers to

interact and select from instances for design evolution. Then, the detail design will be done by designers with human wisdom [18].

The generated images are handled by designers using computer operations, such as rotating, cutting, lighting, coloring and so on. The interactive user interface can be seen in figure 7.

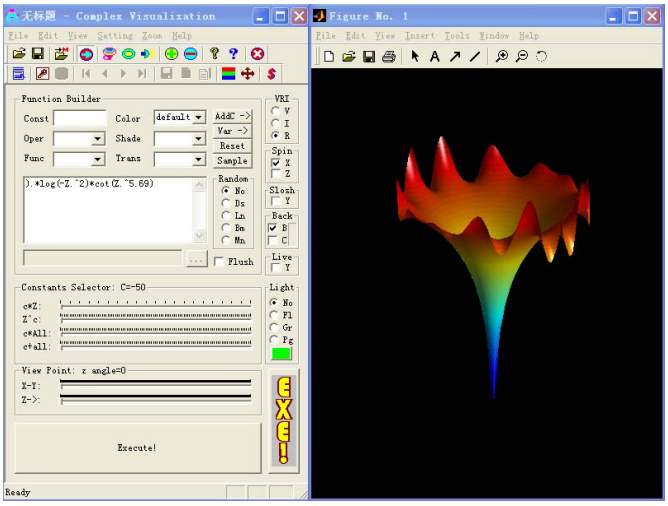

Fig. 7. The interactive user interface

6. Conclusions

This paper presents a novel computer-aided design system which uses tree-like expressions for representing complex mathematical functions and apply this tree structured genetic algorithm in generative design. Although looking simple, the framework employs a feasible and useful approach in a visual evolutionary computing environment. This system facilitates designers in two aspects: 1) diversify instances of design options; 2) enhance the possibility of discovering various potential design solutions.

Designing can be displayed as a dynamic and formal operation of an evolutionary procedure. This study employs the computer as an interface for generating a canonical population for selection. As for generative design, concepts of evolutionary selection are developed that explain different knowledge behaviours. The evolution of populations towards a stable state corresponding to the designers' consensus will be explored in our future work.

7. Acknowledgements

This project is founded by National Natural Science Foundation of China (No. 60970004, No. 60743010) and supported by Natural Science Foundation of Shandong Province (No. Z2008G02).

8. References

1. Innocent T. The language of iconica. In A. Dorin, J. McCormack (eds). First Iteration: A Conference on Generative Systems in the Electronic Arts, CEMA, Melbourne. 1999: 92-104.
2. McCormack, J., Dorin, A. and Innocent, T. Generative design: a paradigm for design research. In Redmond, J. et. al. (eds). Proceedings of Future ground, Design Research Society, Melbourne, 2004.
3. Bentley P..J. Evolutionary design by computers. Morgan Kaufmann Publishers, San Francisco, Calif. 1999.
4. Dorin, A. Aesthetic fitness and artificial evolution for the selection of imagery from the mythical infinite library. In Kelemen, J., Sosík, P. (eds). Advances in Artificial Life, Proceedings of the 6th European Conference on Artificial Life, vol. LNAI2159, Springer-Verlag, Prague, 2001: 659-668.
5. Frazer J. H. etc. Generative and evolutionary techniques for building envelope design .Generative Art 2002.
6. Gero, J. S., Kazakov V. An exploration-based evolutionary model of generative design process. Microcomputers in Civil Engineering 1996; 11:209-216.
7. Soddu, C. New naturality: a generative approach to art and design.Leonardo,2002,35(3):291-294.
8. Holland, J. H. Adaptation in natural and artificial systems. Cambridge, MS: MIT Press, 1975.
9. Sanghamitra, B. Sankar, K. P. Classification and learning using genetic algorithms: applications in bioinformatics and web intelligence. Berlin, New York : Springer, 2007.
10. Gero, J.S. and Peng, W. Assisting Interactions in a Dynamic Design Process: A New Role for an Adaptive Design Tool, in K Hampson, K Brown and P Scuderi (eds), Clients Driving Construction Innovation: Mapping the Terrain, CRC-CI, Brisbane, 2005: 201-210.
11. Soddu, C. Generative art in visionary variations, Art+Math=X conference, University of Colorado, Boulder, 2005.
12. Goldberg, D. E. Genetic algorithms in search, optimization, and machine learning. Addison-Wesley.
13. Koza, J. Genetic programming, on the programming of computers by means of natural selection, MIT Press, 1992.
14. Koza, J. Human-competitive applications of genetic programming. Advances in evolutionary computing: theory and applications. Springer-Verlag New York, Inc. , New York, NY, USA,2003.
15. William, B. L. Koza, J. R. Genetic programming and data structures: genetic programming + data structures = automatic programming!, Kluwer Academic Publishers, Norwell, MA, 1998.
16. Mckay, B., Willis, M. J., Barton, G. W. Using a tree structured genetic algorithm to perform symbolic regression, In Proceedings of the First IEE/IEEE International

Conference on Genetic Algorithm in Engineering Systems: Innovations and Applications, Halifax Hall, University of Sheffield, UK. 1995, 487-498.
17. Liu, H., Tang, M. X., and Frazer, J. H. Supporting Evolution in a Multi-Agent Cooperative Design Environment. Advances in Engineering Software, 2002,33(6):319-328.
18. Liu, H., Tang, M. X., and Frazer, J. H. Supporting creative design in a visual evolutionary computing environment. International Journal of Advances in Engineering Software, 2004, 35(5): 261-271.

Biographical notes: Hong Liu is a Professor of computer science in the School of Information Science and Engineering, Shandong Normal University. She earned her PhD from Institute of Computing Technology, The Chinese Academy of Sciences in 1998. Her research areas of interest include evolutionary computing, computer-aided design, multi-agent system and generative design. She has published in journals such as Computers & Industrial Engineering, Advances in Engineering Software, Applied Soft Computing, Journal of Computer-Aided Design and Computer Graphics and Journal of software.

Received: July 27, 2009; Accepted: October 29, 2009.