Artificial Evolution of Algebraic Surfaces

Edward J. Bedwell & David S. Ebert
University of Maryland Baltimore County
Department of Computer Science and Electrical Engineering
June 1999

Abstract

Procedural techniques are widely used in computer graphics to generate models, textures, and animations. Proceduralism provides a means to abstract a given task away from the user and place the burden of complexity on the computer. Through the use of these procedural methods, developers can provide parametric interfaces to users so that a complex system can manipulated through a relatively small number of simple controls. However, these controls are often non-intuitive and require the user to have some prior knowledge of the underlying techniques. This thesis presents a system that combines implicit surfaces and genetic algorithms in order to solve the problems inherent with parametric control. By treating implicit surface representations as genotypes with a genetic algorithm, complex geometry can be generated in a semi-automated fashion. The only effort required of the user is to select individual surfaces that most closely approximate their target model. Furthermore, the user can bias the mating of two individuals thereby making slight changes to a specific surface in order to refine the model. Thus, the user is provided with a means of procedurally generating highly complex geometry without needing to have any prior understanding of the underlying techniques.

1 INTRODUCTION

Procedural techniques have become an integral part of modern computer animation and modeling packages. Allowing for such effects as realistic plant modeling and automated generation of realistic terrain geometry. A significant benefit of proceduralism is the ability to provide parametric control of these abstract representations to the user. Programmatic tools give the developer a means to provide complex functionality with a relatively small number of input parameters. This provides a means to create tools with simple interfaces that produce relatively complex results. Thus, the amount of time and effort put forth by the user is minimized.

While proceduralism can be extremely beneficial, it also presents some unique disadvantages. Parametric control is a powerful tool, as discussed above. However, providing intuitive parameters can often be problematic. Because the parameters are directly linked to the underlying mechanisms of the technique, the user is often required to have insights into the workings of the procedural tool. This is in direct contradiction with the pursuit of simple interfaces to complex systems.

The system presented in this paper shows that by combining complementary procedural techniques these drawbacks are overcome. We have developed a system that procedurally generates complex geometry through the combination of genetic algorithms and implicit surfaces. The goal is to provide a means of complex geometry generation with minimal time and effort required of the user. Furthermore, the user needs no understanding of the underlying techniques or mathematical complexity to effectively generate interesting, organic, complex forms.

To generate complex geometry in a procedural manner we have chosen to use implicit surfaces as the modeling primitive **[BLO97]**. This representation is exceptionally well suited to proceduralism. The representation is very compact, as opposed to parametric surface definition, and thus is easily manipulated programmatically. However, the problem of complexity becomes immediately evident. To create a surface, the developer must be able to provide the function $\boldsymbol{f}$. This is precisely the disadvantage our system addresses. To do this, the system procedurally explores the set of all functions $\boldsymbol{f}(x,y,z)$ to find a suitable function for the user's needs.

Karl Sims presented genetic programming as means to explore a two-dimensional texture function space **[SIM91]**. As the name implies, genetic programming borrows vocabulary as well as methodology from biology **[HOL75]**. We will treat the implicit polynomial representation of the surface as a genetic code or genotype. The genotype provides directions for building an organism or phenotype. In the system, the phenotype will be the surface represented by the mathematical genotype.

Mating is the process through which the genotypes of one or more phenotypes are combined to create new genotypes, which in turn are used to generate new phenotypes. It is this mating process that allows us to create new shapes from an initial pool of pre-generated parents. The children generated from a mating make up a generation. Each generation must be evaluated and those individuals deemed most fit are chosen to mate to generate the next subsequent generation. The process of choosing the fittest individual is called selection.

William Latham and Steven Todd introduced aesthetic selection based on computer generated imagery **[TOD92]**. Aesthetic selection is when the users chooses the most fit individuals based on subjective criteria, as opposed to automated fitness where some automated function handles fitness ranking and selection. This methodology is particularly useful when it is difficult, or impossible, to quantify the target of the genetic algorithm. Since our system provides a means to interactively search surface space, aesthetic selection is very well suited to our task.

Edward J Bedwell, ebedwe@cs.umbc.edu
David S. Ebert, ebert@cs.umbc.edu

With aesthetic selection as the primary means of control, the user can choose those surfaces exhibiting traits of interest and reiterate the mating process. Hence, the system provides an automated means of exploring the function space, solving the problem of generating the mathematical representation of our target surface. Furthermore, by using a genetic programming approach traits of interest are maintained throughout the evolution of the model. The mating process and sharing of genetic material, in our case pieces of the functional surface definition, facilitates this unique functionality.

Our initial version of the system followed the lead of Sims and Koza by using LISP to handle all the expression manipulation **[EBE98]**. LISP however, does not provide a graphics framework suitable for our needs. Furthermore, the original genetic algorithm that we implemented was very rudimentary and provided very little user control. In the latest version, the system has been rewritten in C++ and significant functionality has been introduced into the genetic algorithm. Mating of surfaces is a more gradual process thanks to a more controlled mating process. Also the user is provided with a weighting mechanism whereby one surface can be favored over another in a given mating. These extensions provide a much more powerful and more controllable system for generating complex implicit models.

1.1 Statement of Purpose

By combining implicit modeling and genetic programming, the system we have developed alleviates the inherent parametric control problem of procedural techniques. The use of the genetic metaphor abstracts the direct manipulation of the geometry away from the user. This abstraction provides a method for modeling complex objects that does not require the user to have any prior knowledge of the underlying techniques used to represent and manipulate the geometry. Furthermore, we have extended the commonly used genetic approach to allow the user a greater level of control through unbalanced mating. Thus, the system provides an intuitive means of generating complex geometry in a way that has not been previously explored, as well as a unique solution to procedural parametric control problems.

2 SYSTEM ARCHITECHTURE

The application is built upon a command line interface giving the user control over all the system's core functionality. The system is initialized by user input of the initial "primitive" parents. These are fed to the parser and the underlying data structures are created. The user then initiates the genetic algorithm, which operates on the parent data structures generating new children genotypes. The new children expressions are sent to the graphics sub-system for display and interaction. Next, based on the phenotypes displayed by the graphics sub-system, the user selects two of the children to become parents. We already have their data structures stored in the genetic algorithm, so they are simply made parents and the process continues.

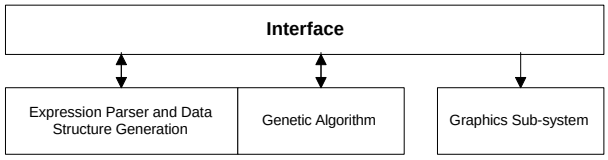


Figure 2-1: General overview of system architecture

2.1 Parsing and Data Structure Construction

The system is initialized by reading in two expressions either provided by the user, or chosen from a predefined list of "primitive" functions. When parsing each expression a tree is built of the sub-expressions based on the precedence of the operators present, as in *Figure 2-2*. A simple binary tree data structure is used to represent this parse tree **[SIM91]**. A binary tree suffices for our purposes because the user is limited to "c-like" expressions consisting solely of unary operators (-), binary operators (-, +, *, /, %), and function calls (e.g. "pow(x, y)"). An initial observation one might make is that function calls do not seem to lend themselves to a simple tree like representation as they may have an arbitrary number of arguments. Furthermore, these arguments should be treated as sub-expressions in their own right. However, as shown in *Figure 2-2*, if the function call itself is considered to be unary operator, and the comma to be a binary operator of the highest precedence, the problem of functions is nicely solved without using an
n-ary tree data structure .

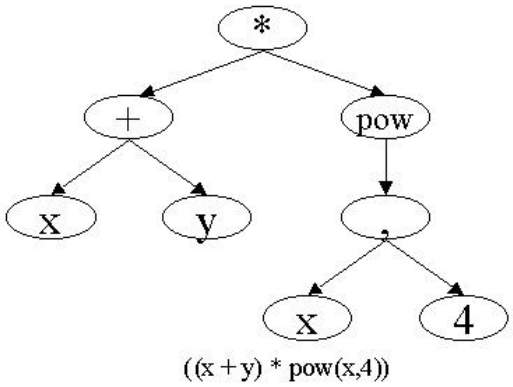


Figure 2-2: Example of a function in a parse tree representation

2.2 Genetic Algorithm

This section will discuss in detail each step of the genetic algorithm used for the surface generation system. We have previously introduced the genetic operations of crossover and selection. The following pseudo-code describes how these operations are applied.

```
parents[ ] = select(finished_kids[a], finished_kids[a])
for each generation
        for (num_children / 2)
                children[ ] = crossover(parent0, parent1)
                        if (random < mutation_probability)
                                mutate(children[0])
                        if (random < mutation_probability)
                                mutate(children[1])
                append children[ ]  to  finished_kids[ ]
pass finished_kids[ ] to graphics sub-system
```

2.2.1 Crossover

The structured representation of the expression tree provides the flexibility that is necessary for the genetic algorithm. As mentioned before, the expression provides the genetic code or genotype. Given the binary tree representation, the tree itself is

viewed as the genotype upon which our genetic operations can be performed **[KOZ90]**. Once the user has defined the initial two parents and the number of matings desired, the system generates children using the crossover mechanism. Crossover, defined as the swapping of genetic material from the parents' genotypes, is applied to the parent trees. One node in each parent tree is selected pseudo-randomly and the sub-trees rooted at these points are swapped. This will generate two new unique trees, or genotypes, containing a portion of each parents' genetic code. The user-defined number of matings will determine how many times this process is repeated. Since the crossover point will be chosen in a random fashion, the children generated from these multiple matings are rarely duplicates.

In order for traits from both parents to be evident in the children, the swapping of genetic information must be somewhat balanced. In other words, no one parent should dominate the provision of genetic material. Choosing the crossover point completely at random does not provide a mechanism for such a balancing. Thus, each node is assigned a value denoting the probability that it will be the crossover point. Since entire sub-trees are swapped, the closer the crossover point is to the root the greater the amount of material swapped. To provide a balance, proportional quantities of each parent must be selected for the sharing. Thus, the crossover probability is defined based on the height of a given node in a tree. The value is determined by $(H_{node} / H_{max} * MAXPROB)$ where H_{node} is the distance of the node from the root, H_{max} is the maximum distance from the root in the tree, and *MAXPROB* is a value between 0.0 and 1.0 denoting the maximum probability that any given node will be the crossover point. With these probabilities in place, nodes closer to the leaves are more likely to be chosen as the crossover point than nodes closer to the root. In addition to providing proportional amounts of the parents' genetic code, this scheme provides a means of limiting the rate of evolution. If *MAXPROB* is defined to be relatively low, the crossover point will be more likely to be found in close proximity to the leaves. Thus, smaller quantities of genetic material are being swapped and change will come about at a slower rate.

In some instances the user may wish to circumvent this proportional balance between the parents. For instance, if one parent surface (phenotype) is very close to meeting the user's target requirements and the other exhibits a trait that the user would like to see incorporated into the first parent. This scenario calls for an uneven swapping of material. The system allows the user to override the default proportional system and define an arbitrary weight to each parent. This is facilitated again through the use of the *MAXPROB* value. Again referring to the above example, defining the *MAXPROB* to be relatively high in the first parent will make it more likely that the crossover point will be located near the root thus providing a large quantity of genetic material. Likewise, defining the *MAXPROB* to be relatively low in the second parent will provide a smaller amount of material. The resulting child will therefore be weighted unevenly in favor of the first parent, while its sibling will, likewise, be weighted toward the second parent.

Throughout the mating process, care must be taken to ensure that the resulting trees still represent valid expressions. Thus, some simple rules must be followed. If one target node is an operator (+, *, %…) the other node must also be an operator. If one target node is a terminal (*x, y, radius…*) or a function call (*pow, sqrt, abs…*) the other node must be a terminal or function call. These two rules ensure that operations stay operations, and nodes that represent some value will continue to represent a value. Finally, under no circumstance can the node selected be a comma. As described above, the "," separating function arguments is treated as a binary operator. However, if we allow a comma to be swapped with another operator the resulting function definition may no longer be valid.

2.2.2 Mutation

In biological systems the genetic operation by which new traits are injected into a population is mutation. Mutation is a modification of a single genotype independent of mating. The purpose of mutation is to bring about traits that are not present in either parent. However, for this to be useful it must be done in a way that preserves the overall structure of the original genotype so as not to completely destroy the evolutionary progress already made. Thus, we have implemented three mutation methods, shown in *Figure 2-3*, that operate in a well defined and controlled manner **[KOZ90]**. Mutation is applied to random children at a fixed probability. If a genotype is mutated, which mutation is applied is also determined randomly based on probabilities defined for each mutation.

First is the sub-tree swap mutation. This is very similar to the crossover process but applied to a single individual as opposed to two parents. This operation selects two nodes at random from the genotype tree representation and swaps the sub-trees rooted at these nodes. This is a non-destructive process that rearranges sub-expressions within the genotype expression. Another possible operation is the node swap mutation. As the name suggests this genetic operation swaps two randomly selected nodes in the tree. The swap node mutation has the highest probability of occurring because it does not modify the tree structure. Finally, the sub-tree destructive mutation selects a node at random within the tree and destroys the sub-tree rooted at this node. This is the least likely mutation to occur because it is the only destructive mutation included in the system.

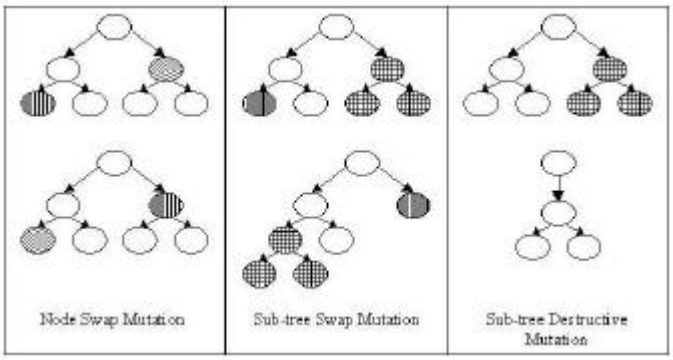


Figure 2-3: Expression tree mutations

The sub-tree swap mutation and sub-tree destructive mutation require that we use the same height based probabilities we used for crossover selection. This is done to reduce the severity of the mutation allowing for gradual change over time. The swap node mutation, however, does not need to make use of the height based probabilities. Since this operation does not modify the organization of the tree there is nothing gained by checking the probabilities. As in the mating process, care must be taken to ensure that the resulting trees still represent valid expression. Thus, the same rules applied to crossover node selection must be applied to the selection of mutation nodes. However, if the mutation is a node swap we must make sure that we do not swap

function calls at all because we cannot guarantee that the proper number of arguments will be provided.

2.3 Graphics Sub-system

Once the user has selected two parents and mated them, the system provides them with a new generation of genotypes. These can be viewed as algebraic expressions, which are arguably of little use. In order for the user to be able to meaningfully evaluate the progeny we must present them with the phenotype for each of the newly created individuals. By performing an in-order traversal of the parse tree the newly created expressions are extracted. These expressions are then passed into the graphics sub-system, where the surface geometry represented by the expression is generated in the form of a polygonal mesh. This geometry is then rendered to provide the user with a visual representation of the phenotype. Furthermore, the system provides the capability to interact with this view of the surface by providing an intuitive interface to rotate, translate and zoom the view of the surface. Thus, the user is able to evaluate the surface in its entirety.

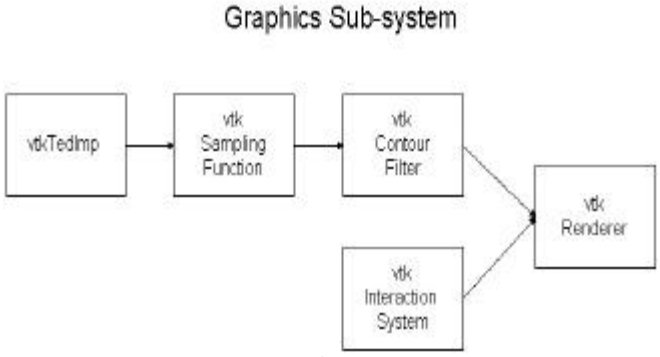


Figure 2-3: Overview of the graphics sub-system

In order to provide this robust graphical interface, the system uses the Visualization Toolkit (vtk) (www.kitware.com). Vtk provides an abstract class definition for an implicit surface (*vtkImplicitFunction)*. To use this class the system defines a concrete sub-class of the *vtkImplicitFunction*. This sub-class defines a method that takes a point in three space as an argument, evaluates a genotype expression at this point, and returns the floating point result. At first glance this may seem like a simple task. However, we must bear in mind that we are generating these expressions dynamically and C++ does not have first order function capabilities. Thus, the system provides this functionality by instantiating an interpreter that can easily evaluate an arbitrary expression. We chose to use TCL for this purpose simply because it is easily integrated into a C/C++ environment, and it provides the function expression capabilities we require.

With the vtk Implicit function in place, vtk provides native functionality for the remainder of the graphics sub-system. The implicit function instance is passed to a 3D-sampling filter. This filter samples a cube, or bounding box, of three space defined by the system to be from –7.0 to 7.0 along the three Cartesian axes. The granularity of the sampling is also an adaptable parameter, which the system defines to be a 50x50x50 regular grid. The values generated by sampling the function space at these locations are then passed on the vtk contour filter. The contour filter interpolates the values to estimate where the function is equal to the system defined threshold value of 0.0. The contour filter then generates a polygonal mesh representing the surface that coincides with these threshold values. This polygonal representation is then rendered using OpenGL. Finally, a window is presented to the user to allow them to interactively view the surface through vtk provided mouse interaction.

To view all the children generated from the genetic algorithm, the system instantiates a separate graphics sub-system for each one. A separate process is forked for each of the children allowing the user to continue the mating process while the rendering takes place. Furthermore, by separating the rendering of each surface into its own process we can take advantage of parallelism in operating platforms with multiple processors.

3 RESULTS

To demonstrate the effectiveness of the system, we have performed a number of experiments. The first supports the claim that the system will produce children that are significantly more visually complex than the initial parents. The experiment detailed in section 3.2 shows that visual traits exhibited by the initial parents are inherited by the children of later generations. The final experiment shows the effectiveness of weighting parents during the mating process.

The implicit functions for all the children shown in the following sections were generated by the previously described system. However, the images were rendered using the POV-Ray (www.povray.org) public domain ray-tracer. We chose to ray-trace the images because the level of detail is significantly higher than that generated by the vtk driven graphics sub-system. The simplified graphics in the system allow for real-time interaction with the generated surfaces at the expense of image quality. Since this document is limited to still images, we have used POV-Ray to generate high quality images. A screenshot of the system-generated graphics is included in Section 6.

3.1 Generation of Complexity

To demonstrate the capability of the system to generate visual complexity, we chose the sphere and torus primitive surfaces as the initial parents. Of all the primitive surfaces provided to the user, these two are visually the least complex *(see Section 6)*. All of the surfaces shown in *Figure 3-1* are individuals in the same family generated by mating a sphere and a torus. All children were generated with a MAXPROB of 0.2, with no weighting.

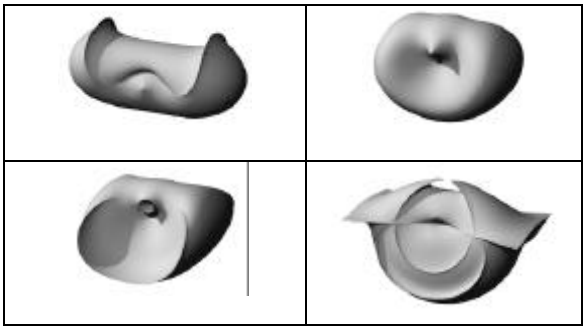


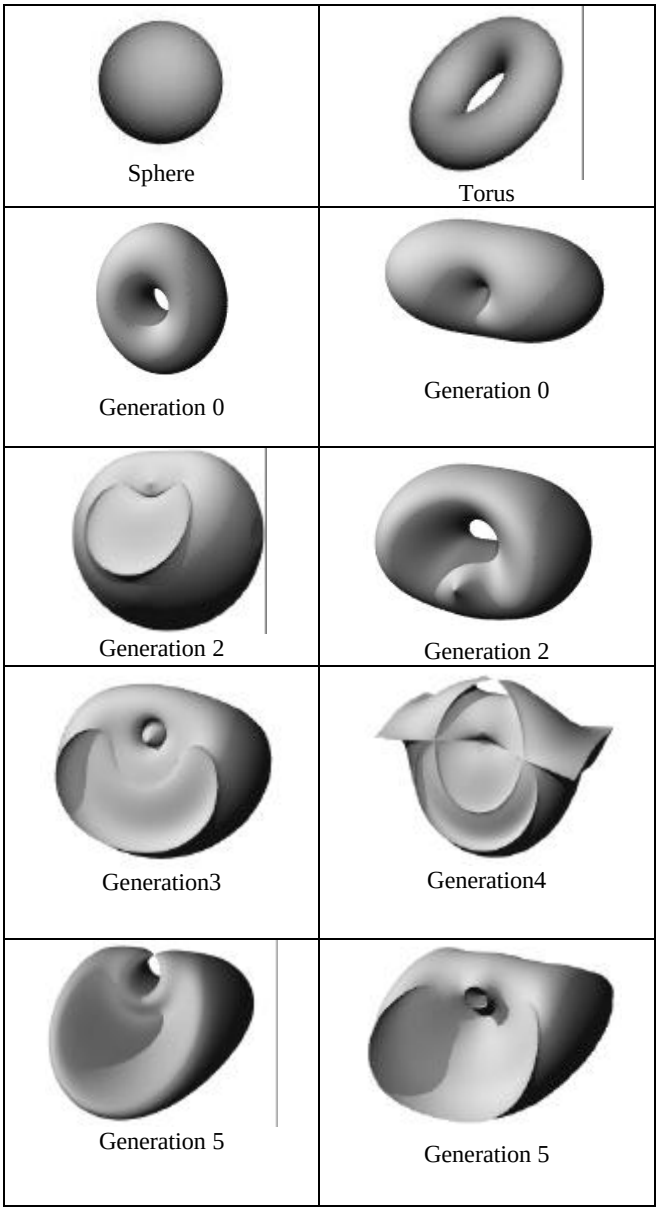
Figure 3-1: Results of complexity generation experiment.

3.2 Inheritance of Visual Traits

This experiment deals with, arguably, the most important function of the system, inheritance of traits from one or both parents. The children below are shown with the initial parents so that the similarities are evident. For the system to be successful, it is necessary to provide a modicum of control. Inheritance, along with weighting, provides a means of directing the evolution toward a general goal surface.

The results shown in *Figure 3-2* demonstrate the inheritance of visual traits throughout the generations created by the sphere and torus mating. Both the hole of the torus, as well as overall spheroid geometry are evident throughout all the generations of the family tree. We have selected a representative sample of the generated surfaces that demonstrate the inheritance particularly well. These results also show that new traits that arise are in turn passed on to later generations. All children were generated with a MAXPROB of 0.2, with no weighting.

.



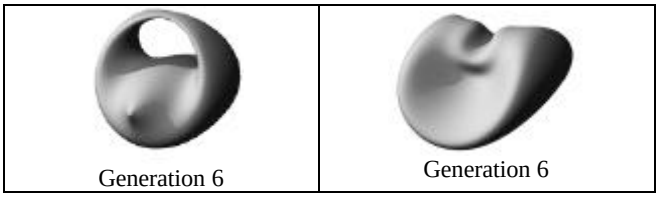


Figure 3-2: Results of visual trait inheritance experiment

3.3 Effects of Weighting

The goal of weighting is to bias the crossover process to maintain the overall structure of a particular parent but introduce a small amount of the other parent. The experiment we conducted used the klein bottle and heart primitive surfaces as the initial parents. We weighted the mating such that the nodes in the heart genotype had their crossover probabilities reduced by 60%, and thus passing only small amounts of genetic material to the mating. The goal being that we could introduce the heart's tail into the overall klein bottle structure.

The results, as seen in *Figure 3-3***,** show that this experiment was indeed successful. The child shown from generation 0 was mated again with the heart weighted at 60% to get the second generation surface showing an extension of the "tail". All children were generated with a MAXPROB of 0.2 before weighting.

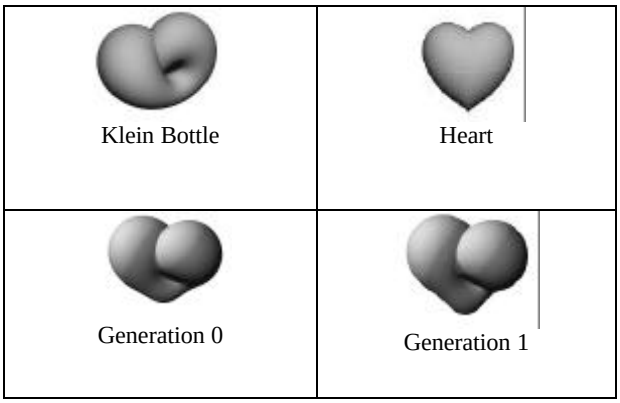


Figure 3-3: Results of the weighted surface mating experiment

4 CONCLUSION

Evolutionary surface generation is a powerful technique. It provides an intuitive method for creating highly complex and interesting geometry. Furthermore, the combination of genetic algorithms with implicit surface modeling provides a unique solution to the problems of parametric control found in many procedural modeling techniques. While the models that the system generates are pseudo-random, the genetic algorithm successfully provides tacit control through trait inheritance. This control allows for a goal-based application of the surface evolution system. Thus, complex target models, conceived by the user, can be approximated without direct user manipulation of the geometry or underlying mathematical implicit representation.

This method of geometry generation is not intended to replace traditional modeling techniques, but rather serves to augment them. For instance, a user of the system can generate complex

shapes, which could be blended with other implicit surfaces. Traditionally, blending surfaces have been limited to spheres and ellipsoid primitives. The evolutionary surface system, however, provides a large variety of other more complex and interesting primitives.

As mentioned above, the system will only provide an approximation of the user's target surface. However, the generated model can then be modified using traditional modeling techniques to more exactly match the target. If the user is generating some complex "organic" form, the automated generation of a similar model will provide a "stepping-off" point that can vastly reduce the time and effort needed to generate the model entirely by hand.

One final application of the system is as an artistic tool. As the results presented in the previous chapter show, very complex and interesting forms can be generated. Digital art is a quickly growing genre to which the system can be directly applied. The genetic algorithm provides a result-oriented means of exploration of all possible implicit surfaces. The process of aesthetic selection makes this exploration an inherently creative process. The individuals generated through the genetic algorithm are as much creations of the user as of the system. One side effect of the mating process is that unbounded or asymptotic functions are also generated. These are of arguably little use as blending primitives. However these often produce stunning geometry and are thus prime examples of the artistic merit inherent in the system. An example of this is shown in *Figure 4-1*.

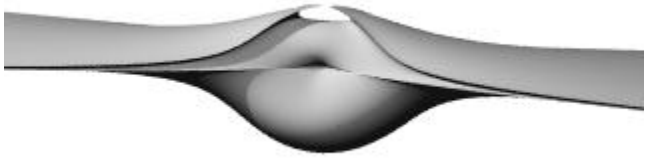


Figure 4-1: An example of an aesthetically pleasing unbounded surface

We believe that the combination of techniques that form this system have significant promise. The state of the art hardware still struggles to generate high quality images of complex implicit functions. The graphics sub-system can produce an average of six images every five minutes on an SGI R10000 Octane workstation. Some of the higher quality ray-traced images shown in Section 3 took nearly twenty minutes each. While five minutes is tolerable for a graduate student doing research, it would need to be significantly faster to be applied in a production environment. However, with the current rate at which computing power is growing, this methodology should be usable in the near future.

In the meantime, the system will be extended to handle blending surfaces in addition to algebraic surfaces. The genetic algorithm can be applied to not only the underlying primitives, but also the blending functions used to build the aggregate surface. Furthermore, blending is often done through functional composition providing a unified mathematical representation of the object. This will facilitate the mating of multiple blending surfaces.

We have shown that complex three-dimensional geometry can be successfully generated in a semi-automated fashion. By combining implicit surfaces and genetic algorithms, the user is provided with a means of intuitively creating complex and interesting forms. Furthermore, these procedural techniques make this technology accessible to those who do not have experience with or knowledge of computer modeling. This system provides more weight to the arguments of those who have carried the banner for procedural techniques.

5 References

[BLO97] J. Bloomental, The Geometry of Implicit Surfaces, in J. Bloomenthal (ed.), Introduction to Implicit Surfaces, Morgan-Kaufmann, 1997, pp. 3-51.

[EBE98] Ebert, D., Bedwell, E., "Implicit Modeling with Procedural Techniques," Proceedings Implicit Surfaces '98, June 1998, Seattle, Wa.

[HOL75] J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor Michigan, 1975

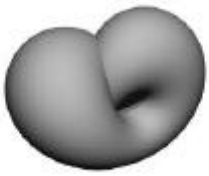
[KOZ90] J. Koza, Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems, Stanford University Department of Computer Science Technical Report STAN-CS-90-1314, June 1990

[SIM91] K. Sims, Artificial Evolution for Computer Graphics, *Computer Graphics*, Vol. 25, No. 4, 1991, pp. 319-328 (Proceedings of SIGGRAPH 91)

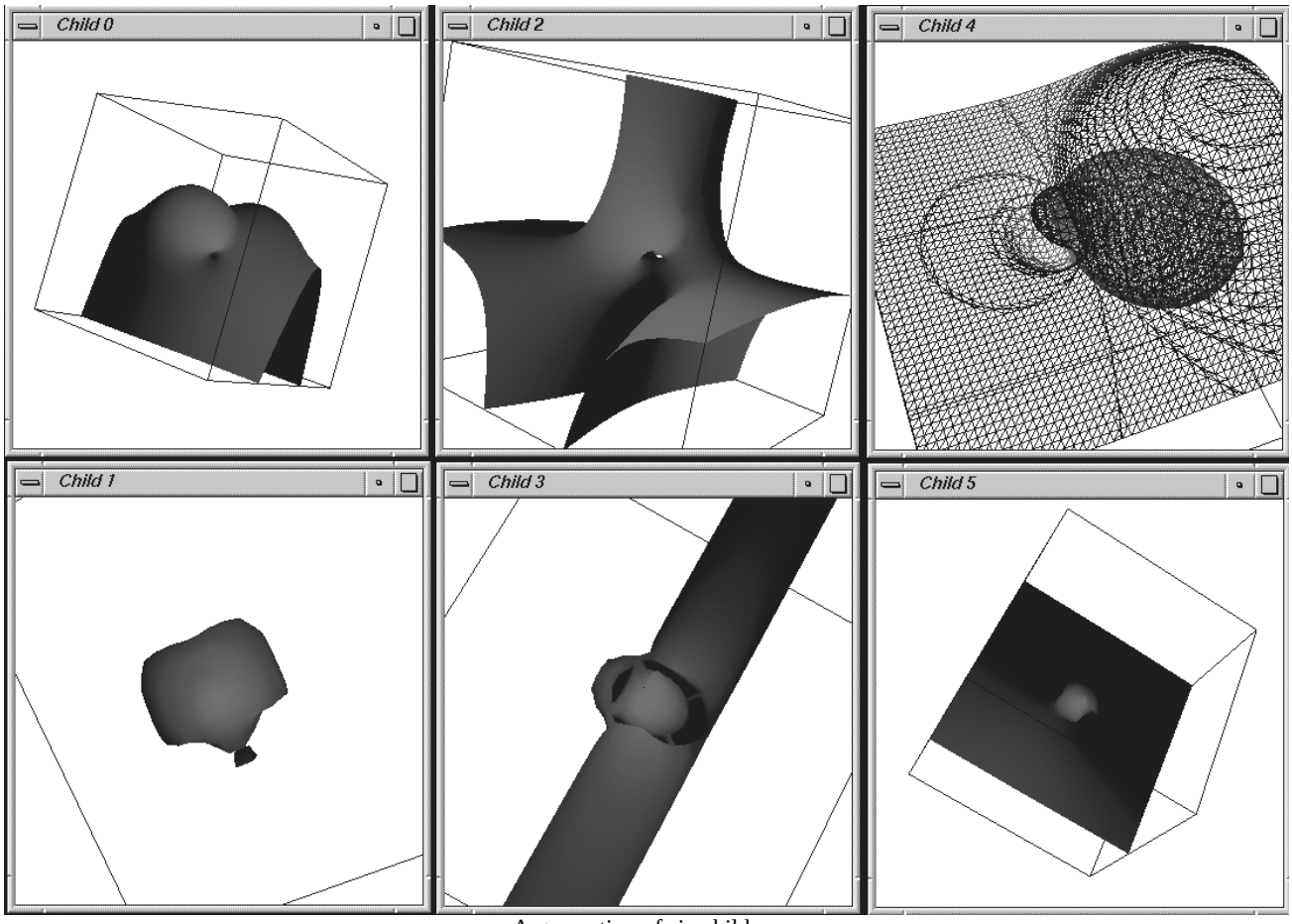
[TOD92] S. Todd, W. Latham, Evolutionary Art and Computers, Academic Press, London, 1992

6 Supplementary Images

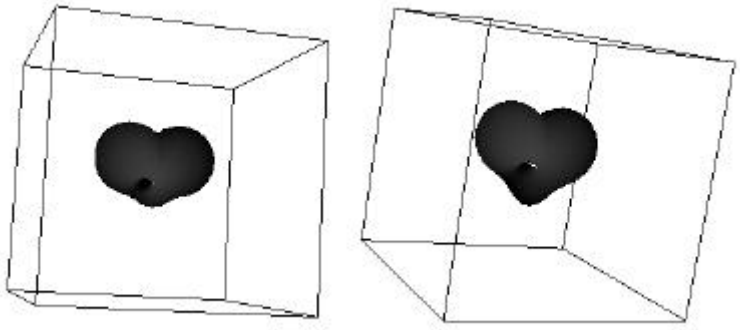
Primitive Surfaces

 Sphere	 Torus	 Cross-Cap
 Heart	 Hunt's Surface	 Klein Bottle
 Steiner's Roman Surface	 Tetrahedral Surface	 Blobby Cube

Graphics Sub-system Images



A generation of six children



An example of weighted mating